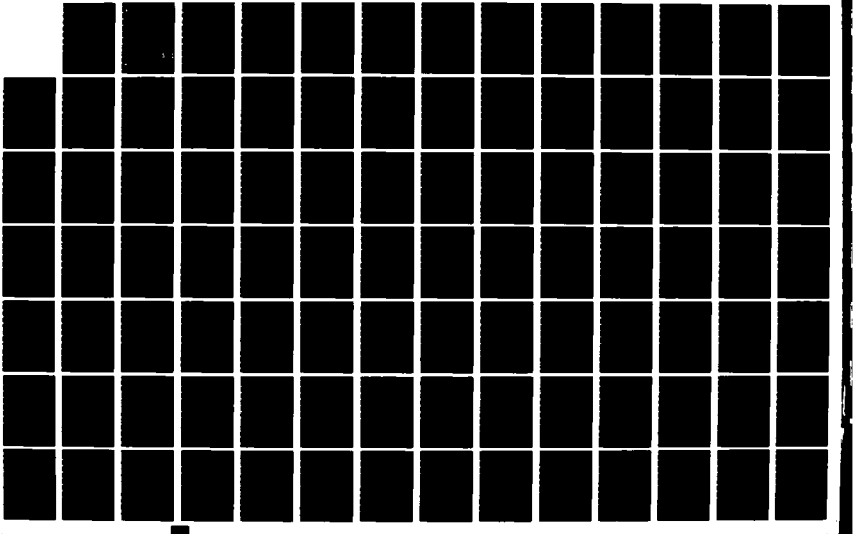
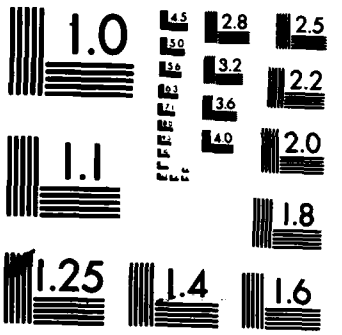
AD-A138 152 DEVELOPMENT OF A COMMUNICATIONS FRONT END PROCESSOR (FEP) FOR THE VAX-11/.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. A F MASTY

UNCLASSIFIED DEC 83 AFIT/GCS/EE/83D-13 F/G 17/2 NL





DEVELOPMENT OF A COMMUNICATIONS FRONT END PROCESSOR (FEP) FOR THE VAX-11/780 USING AN LSI-11/23

THESIS

AFIT/GCS/EE/83D-13

Allan F. Masty
Capt USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/EE/83D-13

DEVELOPMENT OF A COMMUNICATIONS FRONT END PROCESSOR (FEP) FOR THE VAX-11/780 USING AN LSI-11/23

THESIS

AFIT/GCS/EE/83D-13
Allan F. Masty
Capt USAF

Approved for public release; distribution unlimited.

AFIT/GCS/EE/83D-13

DEVELOPMENT OF A COMMUNICATIONS FRONT END PROCESSOR (FEP) FOR THE VAX-11/780 USING AN LSI-11/23

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the degree of
Master of Science in Computer Science

by
Allan F. Masty
Capt USAF
Graduate Electrical Engineering
December 1983

Preface

As complacent human beings, we often take for granted many of the things which mean the most to us. During my degree efforts, I re-awakened to many of the reasons why I married my beautiful wife, Rosanne. Without her loving understanding and deep-felt support, this thesis would not have been possible. She was my prod and support, my counselor and my partner. To her do I dedicate this work.

I wish to thank my thesis advisor, Dr. Gary Lamont and the other members of my thesis committee, Major Walter Seward and Captain John Gordon, for the timely help and direction they provided me. Thanks also to the local "C" language guru, Dr. Thomas Hartrum, who provided invaluable insight into what, at times, appeared to be a demonic compiler.

I also wish to thank my sons, Jeff and Chris. For 18 months, both had to do without some part of me and my time to which they had become accustomed --- only to be replaced by the rantings and ravings of some sort of ogre trapped in the body of an AFIT student.

Accession For	
NTIS GRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

List of Figures

List of Tables

Abstract

A Communications Front-End Processor (FEP) was implemented for a Digital Equipment Corporation (DEC) VAX-11/780 using a DEC LSI-11/23 microcomputer. The LSI-11/23 serviced eight DEC VT-100 terminals and communicated with the VAX-11/780 over an Able Computer Technology, Inc Direct Memory Access (DMA) interface. This investigation proceeded from a FEP design provided in a previous work and culminated in the Telecon 'C' compiler language coding of those design specifications. The design was translated into structure charts defining software module functions and interfaces. Program Design Language (PDL) was then used to outline the processing steps in a structured programming format for each software module. A data dictionary was constructed to document the data and functional module interfaces. The code was implemented in a 'top-down' manner.

CHAPTER 1

INTRODUCTION

1.1 Purpose -

The purpose of this investigation was to design, implement, and test software through which a Digital Equipment Corporation (DEC) LSI-11/23 microcomputer could be configured as a Communications Front-End Processor (FEP) for a DEC Virtual Address Extension (VAX)-11/780 minicomputer within the Digital Engineering Laboratory (DEL) of the Air Force Institute of Technology (AFIT). In its FEP configuration, the LSI-11/23 would provide an interface between the VAX-11/780 and the terminal-related activities of the VAX users.

The LSI-11/23 would service all VAX-bound terminal I/O

interrupts, assembling the input character-by-character until a complete command line had been assembled. The LSI-11/23 would then route this complete message to the VAX-11/780 via a high speed Direct Memory Access (DMA) link. Return traffic to the terminal would be handled in a similar manner. The VAX response (character, line, page, or file) would be sent along the DMA to the LSI-11/23 which would then route it at the proper terminal speed to the user who initiated the activity. The System Physical Device Topology is depicted in Figure 1-1.

1.2 Background -

This investigation is the logical continuation and relies heavily upon material developed in a previous effort [1]. This previous investigation justified the FEP project by establishing it as a cost-effective solution to the resource saturation problem occurring within the DEL VAX-11/780. Resource saturation is the condition of a computer system when it can no longer support additional workload demands placed upon its resources [1:3].

The VAX-11/780 resource saturation condition was shown [1:16] to consist of three components: (1) Overcrowding of

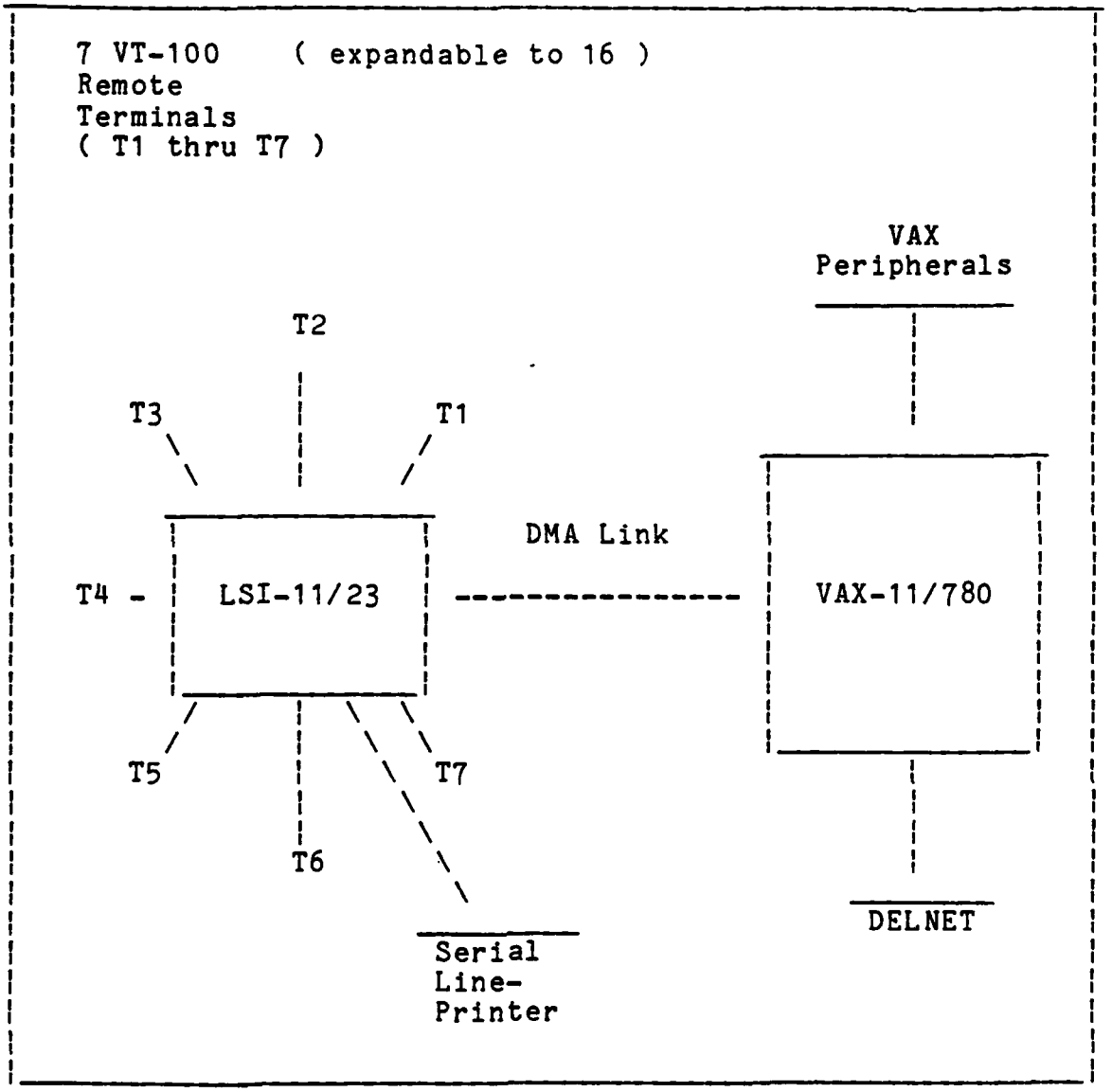


Figure 1-1 System Physical Device Topology

the VAX I/O structure as the number of total system users increased; (2) Increased numbers of pending I/O requests as the number of concurrent users increased; and, (3) Reduced central memory availability brought about by the increased number of concurrent system users. All three components were shown [1:16-18] to be adequately addressed by the FEP concept.

A highly comprehensive Requirements Analysis was then developed [1:18-26] by considering (1) FEP System level Requirements; (2) Hardware Requirements; and, (3) Software Requirements. This latter analysis was performed from the complimentary and increasingly implementation oriented perspectives of the user, the network, and the designer. A summary of the culminating model - the Designer Perspective Software Sub-System Requirements - appears in this report as Table 1-1. The complete model is reproduced [1:94-112] in this report as Appendix A.

1.3 Problem Statement -

The problem pursued during this investigation was to design, code, test, and document the DEC LSI-11/23 portion of the DEL FEP network.

Table 1-1 Designer Perspective Software Requirements

Requirement	Description
1	Local Computer Network
1.1	Two Processors
1.2	Communications Link
1.3	Software On Each Processor
1.3.1	Front-End Software
1.3.1.1	Support User Terminals
1.3.1.1.1	Virtual Link
1.3.1.1.2	Information Routing
1.3.1.1.3	Message Assembly/Disassembly
1.3.1.1.4	Link Assignment Strategy
1.3.1.2	Perform User Tasks
1.3.1.2.1	Operating System Tasks
1.3.1.2.2	Special Functions
1.3.1.3	Comm Link Management
1.3.1.3.1	Control Comm Link
1.3.1.3.2	Assemble Comm Link Message
1.3.1.3.3	Transmit Comm Link Message
1.3.1.3.4	Receive Comm Link Message
1.3.1.3.5	Disassemble Comm Link Message
1.3.1.3.6	Error Check Messages
1.3.2	Host Software
1.3.2.1	Support User Terminals
1.3.2.1.1	Virtual Link
1.3.2.1.2	Information Routing
1.3.2.1.3	Message Assembly/Disassembly
1.3.2.1.4	Link Assignment Strategy
1.3.2.2	Perform User Tasks
1.3.2.2.1	Operating System Tasks
1.3.2.2.2	Special Functions
1.3.2.3	Comm Link Management
1.3.2.3.1	Control Comm Link
1.3.2.3.2	Assemble Comm Link Message
1.3.2.3.3	Transmit Comm Link Message
1.3.2.3.4	Receive Comm Link Message
1.3.2.3.5	Disassemble Comm Link Message
1.3.2.3.6	Error Check Messages

Table 1-1 Designer Perspective Software Requirements (Cont'd)

Requirement	Description
2	Host Operating System
2.1	Multi-Programmed Environment
2.2	Mass Storage
2.3	Comm Link Support
2.4	High Level Language
3	FEP Operating System
3.1	Support for Maximum Terminal Population
3.2	Mass Storage
3.3	Comm Link Support
3.4	High Level Language
4	Consistent User Interface
4.1	Provide "Single User" Environment
4.2	Consistent With VAX/VMS Operation
4.2.1	Single-User/Host Operations
4.2.2	Control/Management Operations
4.2.2.1	Terminal CONNECT
4.2.2.2	Terminal DISCONNECT
4.2.2.3	Command Interpreter
4.3	Procedural Assistance
4.3.1	Single-User/Host Operations
4.3.2	Control/Management Operations
4.3.2.1	HELP Operation
4.4	Easy to Learn and Use
4.4.1	Control/Management Operations
4.4.1.1	HELP Operation
4.4.1.2	Terminal CONNECT
4.4.1.3	Terminal DISCONNECT
4.5	Processing Support Invisible to User
4.5.1	Single-User/Host Operations
4.5.2	Control/Management Operations

Table 1-1 Designer Perspective Software Requirements (Cont'd)

Requirement	Description
5	Operating Environment Compatibility
5.1	Physical Plant Compatibility
5.1.1	Power Source
5.1.2	Temperature Range
5.1.3	Humidity Range
5.2	Academic Compatibility
5.2.1	Unattended Operation
5.2.1.1	Startup Procedure
5.2.1.2	Shutdown Procedure
5.2.1.3	Asynchronous Intermediate Processing
5.2.1.3.1	User Level Messages
5.2.1.3.2	System Level Messages
5.2.1.3.3	Queueing System
5.2.2	Support for 8 Interactive Terminals
5.2.3	Support for Line Printer
5.2.4	Support for Study of LCN
5.2.4.1	Collect Performance Data
5.2.4.1.1	System Level Status
5.2.4.1.2	Terminal Session Statistics
5.2.4.1.3	User Session Statistics
5.2.4.1.4	System Queue Statistics
5.2.4.2	File Transfer
5.2.4.2.1	Transfer To/From Host
5.2.4.2.2	Disk Media
5.2.4.3	Peripheral Sharing
5.2.4.3.1	Route Output To Printer
5.2.5	DELNET Integration
5.2.5.1	Single-User/DELNET Operations
5.2.5.2	Control/Management Operations

Table 1-1 Designer Perspective Software Requirements (Cont'd)

Requirement	Description
6	Supportability
6.1	In-House Maintenance
6.1.1	Hardware
6.1.2	Software
6.2	Expansion
6.2.1	Modular Software
6.2.1.1	Functions
6.2.1.2	Functionally Cohesive
6.2.1.3	Hierarchical Structure
6.2.1.4	Loosely Coupled
6.2.2	Physical Configuration
6.2.2.1	Terminals
6.2.2.2	Processors
6.2.2.3	Comm Links
6.2.3	Inspect Configuration
6.2.4	Modify Configuration
7	Minimum Cost
7.1	On-hand Components
8	Data Security
8.1	No Requirement

1.4 Scope -

This effort was limited to implementing the LSI-11/23 portion of the FEP for several reasons, among which --- time constraints, the level of software risk, and an expanding VAX configuration --- were the most crucial.

It was recognized early that the FEP system would have to operate effectively and dependably in a heavy-use environment. Therefore, careful design, "flawless" coding, and meticulous testing were mandated from the onset to ensure a reliable product. To successfully implement the FEP, a thorough understanding would have to be gained of the VAX/VMS operating system - its services and interfaces as well as its capabilities and limitations. A similarly thorough understanding would be required of the LSI-11/RT-11 operating system. Furthermore, the nature of the DMA link would have to be thoroughly understood. As time passed, it became clear that the pursuit of this thorough understanding for just one of the computers could well occupy the productive efforts of one MS thesis investigation.

Software risk is a measure (not always quantifiable) of the possible exposure to processing corruption introduced by newly created software. The greatest risk applies to long-standing data bases and library files which would have to be re-created from dated checkpoint files and, if

possible, tediously brought up-to-date by recreating the modifications which had occurred since the last checkpoint. The DEL VAX supports several thesis projects currently in progress and departmental Data Base classes. Conversely, there are no users requiring the dedicated use of the DEL LSI-11/23. For the few users who do have occasion to use the DEL LSI-11/23, the possiblity of cross-contamination of user files is non-existant because each user reboots the system at the start of a user session and physically removes their diskettes at the termination of their session. While the LSI-11/23 FEP implementation bears the lesser risk, its realization will benefit the riskier VAX implementation by providing a driver/debugger capability during the VAX FEP development effort.

The final major reason for targeting the LSI-11/23 as the initial FEP implementation was the configuration upgrades that were occurring for the VAX. These upgrades included both hardware and software and would have transformed what was to have been a risky software development effort into a possible perilous effort. These upgrades are projected to taper off with time - thereby restoring a degree of stability to the VAX-11/780 FEP development environment.

1.5 Approach -

The Software Engineering techniques that were used throughout this effort can be classified as "top-down" and "structured". Both of these techniques have proven to be useful [25] during the Software Development Life Cycle of a project. Development tools and documentation aids used throughout this effort include "Structure Charts" (ref Appendix B), a "Data Dictionary" (ref Appendix C), and Program Design Language (PDL).

1.5.1 Software Development Life Cycle -

To better control the development of a project, software managers have identified six separate stages through which software projects pass; these stages are collectively called the Software Development Life Cycle [25:198]:

1. Requirements Analysis
2. Specification
3. Design
4. Coding
5. Testing
6. Operation and Maintenance

1.5.1.1 Requirements Analysis - The Requirements Analysis focuses on the interface between the computer, used as a tool to solve some problem, and the people who need to use it. A Requirements Analysis can aid in understanding both

the problem and the tradeoffs among conflicting constraints, thereby contributing to the best solution [25:199].

1.5.1.2 Specification - While Requirements Analysis seeks to determine whether to use a computer, Specification seeks to define precisely what the computer is to do, but not how to do it. Issues that are examined at this stage include input and output record formats, database layouts, algorithm selections, etc [25:199].

1.5.1.3 Design - In the Design stage, the algorithms called for in the Specifications are developed, and the overall structure of the computer system takes shape. The system is divided into small parts (modules) with constraints as to function, size, and speed [25:200].

1.5.1.4 Coding - Coding is usually the easiest stage. High-level languages and structured programming simplify the task. In one study, Boehm [26] found that 64% of all errors occurred in design, but only 36% in coding. Hamilton and Zeldin [27] report that in the NASA Apollo project about 73% of all errors were design errors. It appears that coding has been mastered better than any other stage of software development [25:200].

1.5.1.5 Testing - The testing stage may require up to half of the total development effort. Testing is divided into three distinct operations: 1) "Module Testing" subjects

each module to the test data supplied by the programmer; 2) "Integration Testing" tests groups of components together; and 3) "Systems Testing" involves the test of the completed system by an outside group [25:200].

1.5.1.6 Operation and Maintenance - These first five stages, collectively forming the development activities, account for only 25% to 33% of the total effort required during the life of the system [25:201]. Maintenance costs ultimately dwarf development costs.

It should be clear that each software development stage may influence earlier stages. The writing of specifications gives feedback for evaluating resource requirements; the design often reveals flaws in these specifications; coding, testing, and operation reveal problems in design [25:202].

It should also become obvious that the particular stage at which an error is detected directly effects the cost of correcting that error. Modifications made to the project have a "rippling" effect that propogates in both directions from the point of change. For instance, customer dissatisfaction with a test result could evolve because the requirements analysis did not precisely describe the customer's desires. In effect, the wrong problem was solved. If this flaw were discovered during the Specification stage, the cost of modification would be

considerably less than it is after coding has been completed.

1.5.2 Software Engineering Techniques -

The Software Engineering Techniques used in developing the software system included Top-Down Design, Top-Down Development, and Structured Programming.

1.5.2.1 Top-Down Design - Top-down design is a technique in which a programmer first formulates a subroutine as a single statement, which is then expanded into one or two of the basic control structures discussed in paragraph 1.5.2.3 (Structured Programming). At each level, the function is expanded in increasingly greater detail until the resulting description becomes the actual source language program. Using this approach, also called "stepwise refinement", the program is hierarchically structured and is described by successive refinements [25:211].

1.5.2.2 Top-Down Development - This is another technique for implementing hierarchically structured programs. Here the top-level routines are written first, and the lower level routines, called stubs, are written to interface with these. The stubs return control after printing a simple message and may return some fixed test value. The stub is eventually replaced by the full module which would then include calls to other stubs. In this manner, an entire

system can be gradually developed and tested [25:212].

1.5.2.3 Structured Programming - A major development in facilitating the programming task is known as "structured programming". The premise here is to use a small set of simple control and data structures. A program then is built by nesting these statements inside each other. This method restricts the number of connections between program parts and thereby improves the comprehensibility and reliability of the program. The "if-then-else", "while-do", and "sequence" statements are one commonly suggested set of control statements for this type of programming [25:211].

1.5.3 Development Tools and Documentation Aids -

Three Development Tools and Documentation aids were chosen from among the many currently available. Selection criteria included a) clarity of presentation, b) user familiarity, c) ease of modification, and d) availability of automated storage and retrieval.

Structure Charts were chosen over Structured Analysis and Design Technique (SADT) diagrams because they were rated higher in "module communication", "training need", and "proliferation" [28:68]. Likewise, Data Dictionary entries were chosen over other database structure representation tools such as Data Structure Diagrams due to the higher degree of maintainability and clarity of expression

[28:72-89] provided with a data dictionary. PDL (pseudocode) was chosen to represent software behavior instead of techniques such as flow charts to take advantage of PDL's clarity characteristics of explicitly representing control structure and nesting level depictions.

1.5.3.1 Structure Charts. - Structure Charts (ref Appendix B) provide a visible and convenient method for portraying the interrelationships between the individual software modules. Hierarchical and scope of control relationships can be easily seen. Also, parameter passing, in the form of data and control flags, between modules can be effectively identified.

1.5.3.2 Data Dictionary - A Data Dictionary (ref Appendix C) is a document in which the names, attributes, and relationships of software data items and functional modules can be described. It serves as a cross-reference locator for the various constants, variables, and procedures appearing throughout the source listings. The functional module entries contain a Program Design Language (PDL) summary of the software logic.

1.5.3.3 Program Design Language (PDL) - PDL (ref Appendix C) is a type of language which contains two structures: "outer" syntax of the basic control statements (ref para 1.5.2.3) and an "inner" syntax that corresponds to the

application being designed. The inner syntax is English statement oriented, and is expanded, step by step, until it expresses the algorithm in some programming language [25:212].

1.6 Overview Of Thesis -

This thesis concentrates upon the software design and implementation of a previously specified FEP [1] system. The Requirements Analysis was accomplished in this previous effort and is summerized in this report in Chapter 2. Chapter 2 also describes several System Design Decisions made during the course of this current investigation. Next, the Network Design and Protocol Issues are discussed (Chapter 3). The thesis continues with the Software Design and Implementation (Chapter 4), the Software Test and Evaluation (Chapter 5), and the Conclusion (Chapter 6).

Appendices include the Software Requirements Analysis (Appendix A), LSI FEP Structure Charts (Appendix B), LSI FEP Data Dictionary (Appendix C), LSI FEP Source Code Listings (Appendix D), LSI FEP Memory Load Map (Appendix E), LSI FEP User's Guide (Appendix F), and LSI FEP Programmer's Guide (Appendix G).

CHAPTER 2

REQUIREMENTS ANALYSIS AND DESIGN DECISIONS

In this chapter, the top level "System Level Requirements" [1:15-26] are examined and then prioritized for implementation. It is from this basic requirements definition and the step-wise decomposition of these requirements that the final "Designer Perspective Software Requirements" (Appendix A) was created. Design Decisions and Trade-offs are then examined and justified.

This chapter also discusses the capabilities and limitations of the LSI-11/23 microcomputer hardware and software in addressing the LSI FEP requirements. This chapter (along with Chapter 3) describes the decisions made during the first 3 Software Development Life Cycle [para 1.5.1] stages (Requirements Analysis, Specification, and Design) of this software project.

2.1 System Level Requirements -

The problem previously investigated [1:5] involved resolution of the VAX resource saturation condition. To solve that problem, eight subordinate requirements [1:18] were identified :

1. Local Computer Network (LCN)
2. Host (VAX) Operating System
3. FEP (LSI) Operating System
4. Consistent User Interface
5. Operating Environment Compatibility
6. Supportability
7. Minimum Cost
8. Data Security

These System Level Requirements and their decomposed sub-groupings are discussed in the following paragraphs. The numbers within the parentheses refer to the system requirements depicted in Table 2-1.

2.1.1 Local Computer Network -

The FEP topology specified a local computer network (LCN) (1) connecting two processors (1.1) with a communications link (1.2). Network software (1.3) would be required for both processors [1:18-21].

2.1.2 Host Operating System. -

Functions required of the VAX operating system include capabilities for supporting a multi-programmed environment

Table 2-1 System Level Requirements

Requirement	Description
1	Local Computer Network
1.1	Two Processors
1.2	Communications Link
1.3	Software On Each Processor
2	Host Operating System
2.1	Multi-Programmed Environment
2.2	Mass Storage
2.3	Comm Link Support
2.4	High Level Language
3	FEP Operating System
3.1	Support for Maximum Terminal Population
3.2	Mass Storage
3.3	Comm Link Support
3.4	High Level Language
4	Consistent User Interface
4.1	Provide "Single User" Environment
4.2	Consistent With VAX/VMS Operation
4.3	Procedural Assistance
4.4	Easy to Learn and Use
4.5	Processing Support Invisible to User
5	Operating Environment Compatibility
5.1	Physical Plant Compatibility
5.1.1	Power Source
5.1.2	Temperature Range
5.1.3	Humidity Range
5.2	Academic Compatibility
5.2.1	Unattended Operation
5.2.2	Support for 7 Interactive Terminals
5.2.3	Support for Line Printer
5.2.4	Support for Study of LCN
5.2.5	DELNET Integration
6	Supportability
6.1	In-House Maintenance
6.2	Expansion
6.2.1	Modular Software
6.2.2	Physical Configuration
7	Minimum Cost
7.1	On-hand Components

(2.1), at least one mass storage device (2.2), the DMA comm link (2.3), and the high level language (2.4) selected for the software implementation [1:21-22].

2.1.3 FEP Operating System. -

Functions required of the LSI operating system include capabilities for supporting a multi-terminal environment (3.1), at least one mass storage device (3.2), the DMA comm link (3.3), and the high level language (3.4) selected for the software implementation [1:22-23].

2.1.4 Consistent User Interface. -

This requirement serves to minimize the amount of "re-learning" required by the VAX user to access the VAX through the LSI FEP. Specific functions include providing all concurrent users with the full spectrum of VAX capabilities available to a single user (4.1) connected directly to the VAX. Furthermore, any special procedures necessary to operate the FEP system must be consistent with the VAX/VMS functional interface (4.2), and a method of obtaining assistance (4.3) on the procedures should be provided. Finally, the user interface must be easy to learn and use (4.4) and the processing necessary to support it should be virtually invisible (4.5) to the user [1:23].

2.1.5 Operating Environment Compatability. -

Physical plant compatability (5.1) requires the FEP to function within the power (5.1.1), temperature (5.1.2), and humidity (5.1.3) ranges existing within the DEL. Academic compatability (5.2) requires the FEP to run unattended (5.2.1) while supporting eight (expandable to 16) concurrent, interactive user terminals (5.2.2) and at least one line printer (5.2.3). As a teaching tool, the FEP should support the study of the LCN environment (5.2.4). Finally, the FEP must be capable of full integration (5.2.5) into the Digital Engineering Laboratory Network (DELNET) [1:24].

2.1.6 Supportability. -

In-house maintenance (6.1), using DEL resources, is required for hardware and software components. Potential system expansion (6.2) of the physical configuration (6.2.1) as well as individual software modules (6.2.2) requires the flexibility provided by current software engineering practices [1:24-25].

2.1.7 Minimum Cost. -

This requirement implies the selection, whenever possible, of network hardware/software components already on-hand (7.1) or readily available to the DEL [1:25].

2.1.8 Data Security. -

No specific requirements were listed under this topic [1:25]. Furthermore, since VAX processing of classified information was not expected to occur, then this System Level Requirement (Requirement 8 in Table I) was eliminated from Table 2-1.

2.2 Requirements Prioritization -

The possibility of not implementing the entire FEP system during the course of one thesis effort was recognized as a highly probable event. Therefore, a priority ranking of the System Level Requirements was accomplished to establish an implementation ordering. This section contains these priority decisions which directly drove the software design and coding efforts.

Those requirements which were constraints included:

a. Most of the Local Computer Network (1) --- the lone exception being the degree of sophistication of the Software on Each Processor (1.3),

b. The Physical Plant Compatability (5.1) portion of the Operating Environment Compatability (5),

c. The In-House Maintenance (6.1) portion of Supportability (6),

d. The Minimum Cost (7).

Other requirements identified capabilities already in existance within the VAX/VMS and LSI/RT-11 operating systems. These included:

a. The Host Operating System (2)

b. The FEP Operating System (3)

These observations left the following areas in which prioritization ranking could occur:

a. The Software on Each Processor (1.3) portion of Local Computer Network (1),

b. Consistent User Interface (4),

c. The Academic Compatability (5.2) portion of Operating Environment Compatability (5),

d. The Expansion (6.2) portion of Supportability (6).

2.2.1 Top Priority. -

The first priority decision made was in the area of Processor Software (1.3). It was decided to concentrate the efforts of this thesis on implementing the LSI portion of the FEP system requirements (ref para 1.4). Host network software would be limited to a driver/debugger application program to manage the DMA traffic at the VAX end of the communications link. This program would initiate DMA

transfers, respond to incoming comm link traffic, and display the comm link traffic upon a VT-100 interactive terminal.

2.2.2 High Priority. -

Certain requirements were viewed as absolutely necessary --- either from a network point-of-view or from an operator point-of-view. Requirements designated as essential fell into one of two categories :

a. If the requirement was a specific function, then it was implemented before any lower priority function,

b. If the requirement described a design strategy, then it served as a design guideline and molded follow-on design decisions.

Requirements identified as high priority include:

2.2.2.1 Provide Single-User Environment. - Of all the System Level Requirements, this one most closely relates to the overall reason for the FEP concept --- namely to solve the VAX resource saturation problem. This requirement basically means that each concurrent user will experience an environment in which they could imagine that they were the sole user connected directly to the VAX. Implied within this definition are several important concepts:

A. Reliable Terminal/Process linkage in both directions. This requires the FEP software to communicate the terminal requests to the proper

process executing on the host and to return the proper, uncontaminated response from the host to the appropriate terminal.

B. Terminal response delay should not be noticably longer during FEP operations than the delay which could reasonably be expected to occur during single-user operation. This requires fast executing code and rapid movement of data.

C. The full repertoire of VAX/VMS commands must be available to each concurrent user without adding any FEP-unique command overhead to his use of the FEP. This implies a processing transparency requirement for the LSI FEP software.

2.2.2.2 Consistent With VAX/VMS Operation. - This function implies that all services normally provided by the host's terminal device drivers must now be provided within the LSI FEP so that the VAX/VMS would be presented with a Consistent User Interface.

2.2.2.3 Unattended Operation. - Since the host VAX operates in an unattended configuration, its LSI FEP extension should be designed to not require any additional manning overhead.

2.2.2.4 Support for 7 Interactive Terminals. - Support for 8 Interactive Terminals (ref para 2.1.5) could not be met due to system limitations. The LSI-11/23 hardware components reside within a Plessey Peripheral Systems MICRO-II based computer system [2:1-3]. The Plessey PM-MFV11A Multifunction board provides four Electronic Industry Association (EIA) RS232 ports which may be used for

console device interface, printer, modem, or spare terminals [3:1-1]. Port 1 is permanently assigned to the console, leaving only 3 ports on the PM-MFV11A card for spare terminals. A DLV11-J card was inserted into the LSI-11 bus to provide 4 additional asynchronous serial interfaces (ports) to the FEP. These two cards provide a total of only 7 (not 8) interactive terminal interfaces. Therefore, requirement 5.2.2 (Table 2-1) was changed to reflect the current physical limitation of Support for 7 Interactive Terminals.

> [NOTE: Although not designed for this purpose, the console can be used as the eighth terminal and software support for this contingency was written. However, all LSI FEP system error and status messages will be output to the console screen. This could become quite annoying to an interactive user stationed at this console.]

2.2.3 Medium Priority. -

Assigned to this category were functions which were desirable for early implementation, but not required for the basic FEP to operate. These functions could have been deferred for implementation in a follow-on thesis investigation if time ran out during the current thesis effort. Medium priority functions include:

2.2.3.1 Easy to Learn and Use. - This function specifies that the user not be burdened with additional operating overhead in order to communicate his requests to the VAX. This requirement is consistent with the high priority requirement to Provide a "single-user" Environment (ref para 2.2.2.1). With the successful implementation of this high priority requirement, it is expected that a minimum operating overhead will fall out as an inevitable by-product.

2.2.3.2 Processing Support Invisible to the User. - This requirement specifies that the user need not know where, within the network, his requests are processed. It is absolutely transparent (and of no importance) to him that terminal device driver/interrupt service functions, for example, have been removed from the VAX processor to the LSI-11/23 processor. As with the previous medium priority requirement, it is expected that Invisible Support to the User will be satisfied during implementation of the high priority Provide a "Single-User" Environment requirement.

2.2.3.3 Support for LCN Study - This requirement specifies the periodic trapping of system queueing data for later, off-line reduction and analysis. Although providing a significant opportunity to monitor network status in a dynamic environment, this requirement's implementation is not essential for the FEP system to function.

2.2.3 " Expansion. - This requirement, which is decomposed intc Modular Software and Physical Configuration specifies a flexible implementation which would easily accomodate system modifications such as the addition of more terminals or software function/subroutines. This requirement was assigned a medium priority primarily because it's implementation conflicts, at times, with that of the higher priority requirements. For instance, modular software engineering practices encourage a minimum of data-structure sharing between software modules. This concept requires each module to define the data items and structures that will be required during its execution. On the LSI-11/23 these locally defined data items are created each time the subroutine is called and destroyed each time the subroutine exits. Although well-designed from a software engineering perspective, this repeated creation and destruction of the same data definitions is counter-productive in a real-time, interrupt-driven application. The additional processing overhead required to isolate modular data creates an unacceptable delay in the network processing, especially those functions that service interrupts. For these reasons, the Expansion requirement has been assigned a medium priority so that implementation conflicts may be resolved in favor of the higher priority tasks.

2.2.4 Low Priority. -

Requirements assigned in this category include those which are desirable, but less important than even the medium priority tasks. Included within this category are:

2.2.4.1 Procedural Assistance. - This function specifies the need for "HELP" commands to be made available for the user to operate the network. It is envisioned that, if the requirements for "Single-User" Environment (ref para 2.2.2.1) and Easy to Learn and Use (ref para 2.2.3.1) have been properly implemented, then the requirement for a "HELP" file will be near non-existent. Therefore, a decision as to the need for its implementation should be deferred until after the other two requirements have been implemented.

2.2.4.2 Support for On-Line Printer. - This requirement, part of the original network design [1:35], specified a serial line printer to be connected to the LSI. However, the current effort is directed at implementing the basic FEP system and then enhancing it as resources permit. Under this concept, moving the printer from the VAX to the LSI becomes an enhancement rather than a high priority requirement. The driving goal during the early implementation stages is to free the VAX of the interactive I/O overhead by relocating the terminal device

driver/interrupt routines to the LSI. To move the printer functions concurrently would be to expose the initial FEP implementation to a higher risk factor than is clearly warranted at this early stage. Delaying the printer relocation will also simplify the test and evaluation stage of the initial implementation.

2.2.4.3 DELNET Integration. - This function pre- supposes the existance of the DELNET. Pending the implementation of the DELNET, this requirement will be assigned a low priority status.

2.3 Design Decisions And Tradeoffs -

Several situations requiring major design decision trade-offs were encountered during this investigation. These were considered "major" design decision trade-offs because their resolution affected and guided further design issues. Throughout the FEP effort, additional "minor" trade-off decisions were made which influenced the implementation of specific functions (i.e. size of buffers and headers, etc.). Discussion of these "minor" issues is deferred until the functions themselves are discussed. These "major" issues are discussed in the following paragraphs.

2.3.1 Execution Speed vs Program Size. -

This is a familiar trade-off decision often faced by the application programmer. In most applications, a program can be made to execute faster only at the expense of increased memory requirements. Likewise, memory may often be saved by recoding the program to execute slower. Rarely will a programmer be able to concurrently optimize both program parameters.

The FEP system is no different than most real-time applications. It was expected to function in a dynamic environment, servicing a variable number of randomly arriving "bursty" terminal requests. This environment would be best served with a fast executing program which would minimize the chance for FEP saturation (and its resulting loss of data) when the system was pushed to its maximum activity levels.

Supporting this "speed" over "size" decision was the availability of the RT-11 Extended Memory (XM) Monitor [7:4.1]. The XM monitor provides a usable memory capacity four times that of the standard RT-11 Single Job (SJ) Monitor [7:4.6]. Paragraph 2.3.7 describes how the RT-11 monitor was to be utilized.

2.3.2 Interrupt Structuring. -

It was recognized early that all incoming data traffic to the LSI-11/23 would have to be serviced by honoring an I/O interrupt. This was required because the LSI uses a single memory location, the Receive Data Buffer (RBUF), for receipt of character data. Ensuing characters will over-write the previous character regardless of whether that previous character has been retrieved by an application or systems program. If the RBUF for each terminal was serviced synchronously, then valuable processing time could be wasted checking ports at which no new data had arrived since the prior servicing.

On the other hand, interrupt-driven Input servers would only execute when needed and could be constructed to contain the minimum amount of processing needed to fetch the character, store it away, and perform mimimal immediate response for a select subset (i.e. --- backspace, carriage return, control-C, etc.) of the possible character set.

Similar reasoning would seem to have advocated an interrupt-driven output server as well. However, three important considerations suggested synchronous servicing of output character traffic as a better approach.

a. Although the receipt of input data was random in nature, the output of data traffic would be totally under program control.

b. Like the RBUF, the Transmit Data Buffer (XBUF) could be overwritten with ensuing characters if the serial interface was not provided a sufficient time interval during which it could transfer the previous character.

c. It was considered highly likely that incoming data interrupts could occur while the LSI was busy executing an output interrupt routine.

This last concern presented two possible treatments:

a. Disable interrupts while processing an interrupt.

b. Allow nesting of interrupts.

Both approaches presented drawbacks. Locking out interrupts would present a high probability for lost data because the RBUF of a fast keyboard typist could be overwritten before the LSI was notified of a new character's presence there. Nesting of interrupts would slow down the processing due to the overhead involved in storing and retrieving register information required for the orderly resumption of an interrupted routine. Nesting of interrupts would also require the additional overhead of designing, coding, and testing re-entrant subroutines.

Both of these problems can be avoided by implementing a synchronous servicing discipline for outbound characters from the LSI. A polling method could be implemented in which the Transmit Ready bit in the Transmitter Control and Status Register (XCSR) would be interrogated prior to the

program moving the next character into the XBUF. This bit is set by the hardware when the serial interface is ready to accept the next character [20:507]. Usually, polling would consist of the software executing a tight loop until the bit was set. For the FEP system, any loop idling for indefinite time durations would have to be minimized. Conveniently, however, at least one indefinite loop is mandated for the system. This loop is the large idle-loop that encompasses all non-interrupt processing functions. This loop will be entered after system initialization and iterate continuously until the system is terminated. The polling routines were selected for inclusion within this large loop.

If the Transmit Ready bit in XCSR was set when checked, then the next output character would be moved to XBUF. If the bit were not set, then the next synchronous task would be executed and a recheck of XCSR would be delayed until the next pass through the loop. Since most iterations of the large loop would not find new work, it was projected that this scheme would introduce very little additional delay in the movement of output characters to the target device. On the other hand, all XCSR/XBUF pairs could be interrogated within the same loop, thus economizing the wait process when output data was available for more than one device.

2.3.3 Device Handlers vs. Interrupt Service Routines. -

Device Handlers (drivers) are routines that provide the interface to the computer hardware devices. The handlers drive, or service, peripheral devices and manage information transfer between memory and the devices [7:2.19]. Device handlers are usually stand-alone programs which must be loaded into physical memory before they can be used. An interrupt service routine, on the other hand, is coded as an integral part of the application program. This routine is called by the main program to initiate an I/O transfer. The routine then returns execution control back to the application program until the transfer is complete --- at which time the device issues an interrupt. This interrupt forces a transfer of execution control back to the interrupt service routine which can then take appropriate action (restart the I/O transfer, return to the program, or possibly retry the transfer in case of an error) [7:6.2].

The major advantages in using Device Drivers are that they:

A. Provide device independence for the application program

B. Can share processor time with other processes

C. Are simple to use

The major advantages in using In-line Interrupt Service

Routines are :

A. Their speed of execution

B. The amount of control information they provide

Device independence was never a design consideration because DEC VT-100's were specified [1:34] as the interactive terminal devices. Since no other process was expected to be simultaneously executing on the LSI-11/23, there was no requirement to ensure process-sharing of resources. Simplicity of use was not, by itself, sufficient justification for using Device Drivers. On the other hand, speed of execution was a critical component for real-time applications and supported the high priority requirement to "Provide a 'Single-User' Environment" discussed previously (ref para 2.2.2.1.B). Providing additional control information was a benefit whose implementation cost was negligible. Therefore, the third "major" design decision was to implement In-Line Interrupt Service Routines rather that Device Handlers.

2.3.4 Device Priority. -

Specifying the device priorities amounts to determining the servicing order when simultaneous interrupts occur. Although the LSI-11/23 allows four interrupt levels, it was decided not to utilize this feature due to the interrupt nesting problems that could occur [20:178-180 and

para 2.3.2]. Instead, device card placement on the LSI UNIBUS would be used to enforce device priority. When devices of equal priority level request an interrupt, priority is given to the device electrically closest to the processor [20:350]. Since the overall goal of the FEP is to reduce the VAX workload, then DMA traffic could not be allowed to back-up within the host. Therefore, the DMA device was selected as the highest priority device, the terminals next, and the System Operator Console (SOC) last.

However, system constraints prevented this exact ordering from being implemented. The UNIBUS is contained within the Plessey chassis. The Plessey orders the priorities of the 9-slot UNIBUS backplane slightly differently. The PM-MFV11A multifunction board is allocated first priority before any other cards. This means that the System Operator's Console (SOC) port and the other 3 serial I/O ports on that card will have a higher interrupt priority than the DMA card. The 4 port serial I/O interface card, DLV11-J, will still be inserted further away from the processor than the DMA card. Those software functions whose servicing order is arbitrary will be coded to discriminate in favor of rapid movement of DMA traffic. (These and other items of Software Design are discussed in Chapter 4).

2.3.5 Virtual Mapping vs. Privileged Mapping. -

Mapping is the process of associating Virtual Addresses with Physical memory locations [7:4.18]. A Virtual Address is a value in the range of 0 through 177777 (octal). It is a 16-bit address within a program's 32K-word address space --- created at assembly time and modified during the linking process. A Physical Address is the actual hardware address of a specific memory location. In the RT-11 Extended Memory (XM) environment, Physical Addresses lie in the range 0 through 777777 (octal) because the XM Memory Management Unit (MMU) appends two additional high order bits to the 16-bit Virtual Address to create a Physical Address space of 128K words.

Virtual Mapped jobs load into memory at offset 500 (octal) from the start of the user address space [7:4.26]. Since all jobs can only access memory addresses within their user address space, virtual jobs cannot access addresses 0 through 477 (octal). Privilege Mapped jobs load into memory at offset 0 and can access all of memory [7:4.27]. Physical Addresses 60 through 477 (octal) are used by the RT-11 monitor for interrupt vector linkages. These vectors are loaded with the addresses of the interrupt service routines which are to assume execution control upon device issuance of an I/O interrupt.

Since a previous design decision (ref para 2.3.3) indicated the preference for In-line Interrupt Service Routines over Device Handlers, it becomes necessary to be able to access these interrupt vectors and load them with vector information. Since Virtual Mapping cannot access these locations, Privileged Mapping is required.

A memory load map of the final LSIFEP program mapping is included as Appendix E.

2.3.6 Processor Transition to System State. -

The XM monitor can support multiple programs, although only one can be actively using the processor at any given time. The non-active jobs may have been placed in their current wait state for various reasons: blocked awaiting I/O completion, pre-empted by a higher priority job, hybernating for a pre-determined period, etc.

When the RT-11 scheduler commands the processor to run a different job, the monitor executes a Context Switch [7:3.23]. Context switching is the procedure through which the monitor save's a job's context - its machine environment and important job-specific information - and begins execution of another job.

The following information is saved in a context switch [7:3.29]:

a. Processor Status Word (PS)

b. Program Counter (PC)

c. Stack Pointer (SP)

d. Registers R0 through R5

e. Kernal Page Address Register #1 (PAR1)

f. Memory Management fault trap vector

g. Break Point Trap (BPT) vector

h. Input/Output Trap (IOT) vector

i. TRAP vector

j. System Communication Area (locations 40-52)

k. Floating Point Processor (FPP) registers

l. FPP status word

m. Stack

n. Impure data area

Applications programs execute in User State, during which time Context Switching is enabled. However, the monitor forces a transition to System State whenever it determines that a potential Context Switch must be delayed. Typically, these situations occur when the monitor is executing and is modifying important data structures. The System State ensures that no other application program can interrupt the monitor, gain execution control of the processor, and contaminate the data structure modification process before it has completed [7:3.24].

Any I/O interrupt issued while the system is in System State will be delayed until User State is re-entered. This could present serious consequences for the FEP system. The VAX-to-LSI DMA transfer would function as a high bandwidth input from an independent source. It is conceivable that data could be lost and error conditions not be recognized immediately if these interrupts are delayed for too great a period of time. For this reason, it is imperative to limit the number of transitions to System State as well as the duration of those System State processes which come under programmer control. These include certain types of I/O transfers, programmed requests (.PROTECT, .CHCOPY, .INTEN, etc.), and XM mapping requests issued from within the program [7:3.24].

2.3.7 RT-11 Extended Memory (XM) Utilization. -

Low Memory is the physical memory between 0 and 28K words (addresses 0 to 177777). Extended Memory is the physical memory above the 28K word boundary (addresses 200000 to 757777) [7:4.1]. The topmost 4K words (addresses 760000 to 777777) form the I/O Page and are reserved by the operating system for register usage and I/O transactions. Portions of the FEP software reside in Low Memory along with all of the RT-11 XM Monitor software. Other portions of the FEP software are mapped to Extended Memory with the aid of the Memory Management Unit (MMU) [7:4.8].

There are two ways [7:4.4] to map portions of a program to Extended Memory. One is by issuing XM programmed requests from within the program itself. The other way is to generate the entire executable program image at link time by explicitly directing the linker to map specific segments to Low Memory and others to Extended Memory [9:11.4]. This latter approach was selected because it allowed more flexibility and speed of change when such changes were required in the mapping assignments. Furthermore, this decision supports the need to limit the number of programmed requests (ref para 2.3.6) issued by the LSI FEP application software.

Certain constraints are forced upon the mapping process by the LSI hardware and software architectures [7:4.8]:

a. Interrupt service routines must be located entirely within the low 28K words of memory.

b. Interrupt service routines must neither reside in nor reference addresses within the range of 20000 through 37777.

With these restrictions in mind, all software processing logic for the LSI FEP was intended to reside in Low Memory. All buffers of significant size would have been allocated to Extended Memory. This isolation of data from instructions would have allowed independent "tweaking" of either without causing massive readjustments in the other to

accomodate the change.

However, during early implementation and testing, another (undocumented) constraint was discovered which precluded this assignment scheme. Apparently, in addition to residing entirely within low memory, an interrupt service routine can only reference data locations within lower memory. Attempts to write to buffer areas in extended memory from within low memory interrupt service routines resulted in low memory instructions being overwritten with data.

Since these buffer areas had to be allocated to low memory, a real dilemma developed as to exactly which program functions could be removed to extended memory. The final discriminant was whether the function could potentially be called by an interrupt service routine. The eventual memory assignment strategy is discussed in Chapter 4.

2.3.8 High Level Language Selection. -

Two high level languages were implemented for use on the LSI-11/23 in the DEL: NBS Pascal and TELECON 'C'. One inherent limitation with Pascal is that address selection is accomplished by defining pointers over whose values the programmer had no control. Also, the setting of interrupt vectors would be impossible in NBS Pascal because the address of the vector could not be specified by the

programmer. A third NBS Pascal limitation is that the address of a subroutine could not be fetched or moved.

These limitations could be circumvented by implementing these critical functions at the assembly language level, but this approach would detract from the high maintainability and expandability requirements already specified (ref para 2.1.6). The 'C' programming language [21, 22] possessed none of these limitations. Therefore, it was chosen as the implementaion language for the LSI FEP.

Although another 'C' compiler product was available from Whitesmiths [29, 30], it had not been received by the DEL prior to the implementation phase of this investigation. For this reason, the Telecon 'C' compiler was chosen for source program compilation.

2.4 Summary -

This chapter described the system level requirements for the LSI FEP. It then prioritized these requirements to facilitate an orderly design and coding phase. These priority categories included top priority, high priority, medium priority, and low priority. The chapter concluded with an examination of the design decisions and trade-offs

which occurred due to conflicting constraints and LSI-11/23 capability (or inability) to adequately address the system requirements.

CHAPTER 3

NETWORK DESIGN AND PROTOCOL ISSUES

This chapter describes the design of the communication network. In order to clarify and resolve the network issues, the system topology will be studied from a network node perspective rather than from the physical device perspective of Figure 1-1. These issues are discussed in terms of physical and logical nodes, which are defined and then identified. The chapter then continues with a presentation of the protocol requirements at the various network layers. The FEP System Network Node Topology is contained in Figure 3-1.

3.1 Logically Connected Nodes -

This type of node represents a software module which executes as an autonomous, specialized function within the

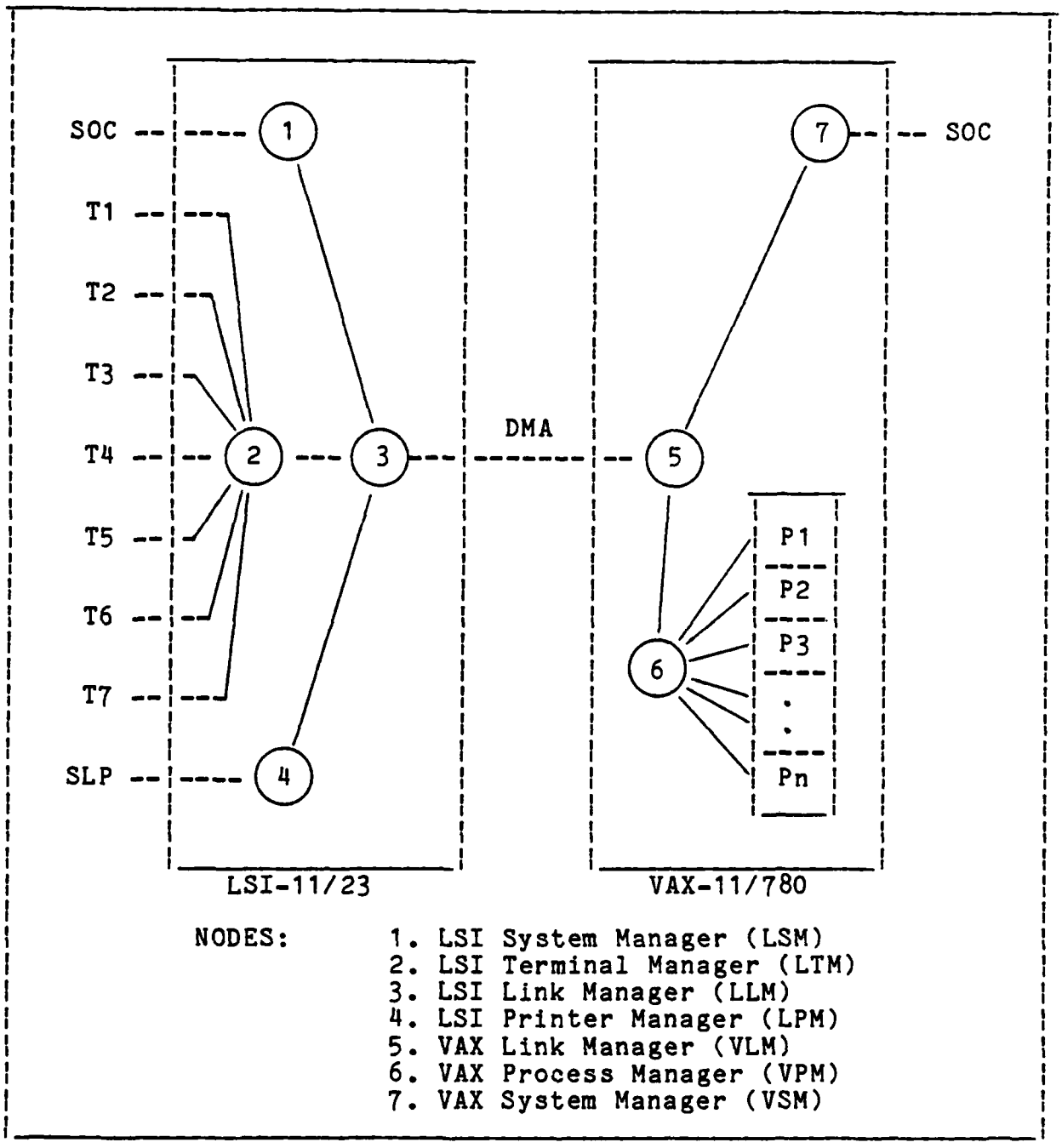


Figure 3-1 System Network Node Topology

network. Data is moved between logical nodes using a packet switched [5:116] concept. Seven nodes were identified:

3.1.1 LSI System Manager (LSM) -

LSM Executes the interrupt routines and other functions to manage the LSI System Operator's Console (SOC) through which is exercised operator control over the LSI system. The LSM controls the SOC/LSI interface.

3.1.2 LSI Terminal Manager (LTM) -

LTM Executes the interrupt routines and other functions to manage the seven User Terminals (T1 - T7). The LTM controls the TTx/LSI interface.

3.1.3 LSI Link Manager (LLM) -

LLM Executes the interrupt routines and other functions to manage the LSI end of the LSI/VAX interface.

3.1.4 LSI Printer Manager (LPM) -

LPM Executes the interrupt routines and other functions to manage the LSI Serial Line Printer (SLP). The LPM controls the SLP/LSI interface.

3.1.5 VAX Link Manager (VLM) -

VLM Executes the interrupt routines and other functions to manage the VAX end of the LSI/VAX interface.

3.1.6 VAX Process Manager (VPM) -

VPM Executes all functions required to interface DMA-delivered traffic with the VAX/VMS Process Management, Memory Management, and other system functions. The VPM is the only logical node which does not service an I/O device.

3.1.7 VAX System Manager (VSM) -

VSM Executes the interrupt routines and other functions to manage the VAX System Operator's Console (SOC) through which is exercised operator control over the VAX system. The VSM controls the SOC/VAX interface.

3.2 Physically Connected Nodes -

Physically connected nodes are logical nodes which are connected to each other by means of a physical data link of some type. The possible data links include: a) Serial; b) Parallel; and c) DMA links. No requirements have been defined for parallel data transfers within the FEP system.

The physical data paths within the FEP system include: a) Peripheral/LSI interface; b) LSI/VAX interface; and c) Peripheral/VAX interface. Of these three, only the LSI/VAX interface connects logical nodes (LLM and VLM) at both ends. Therefore, these two nodes are the only physically connected nodes in the system.

3.3 Protocol Layers -

The discussion of protocol layers begins at the lowest (Physical) level and progresses upward using the ISO Reference Model [5:15]. This model is included as Figure 3-2. The network issues at each level are examined and applied to the FEP/VAX environment.

3.3.1 Physical (Layer 1) -

Issues normally addressed at the Physical level include Multiplexing [5:103], Terminal Concentration [5:122], Packet Assembly/Disassembly [5:122], and Error Control [5:125-132].

3.3.1.1 Terminal Multiplexing - Terminal Multiplexing involves using a device (terminal controller) that accepts input from a collection of lines in some static, predetermined sequence and outputs the data onto a single output line (the DMA link) in the same sequence. Each

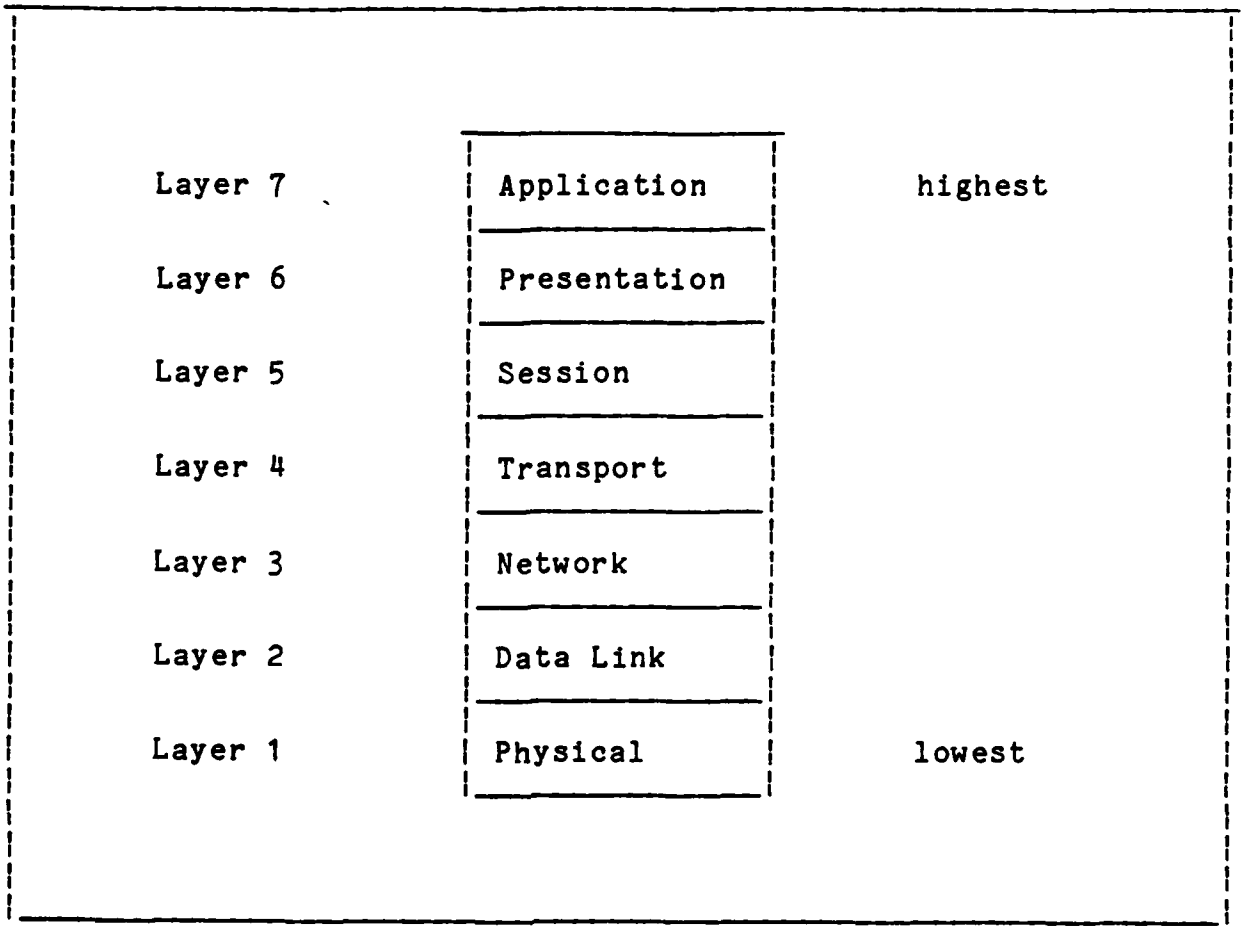


Figure 3-2 ISO Reference Model Protocol Layers

terminal device is assigned a time slot in which only its traffic may be transmitted along the output line. This output line must have the same transmission capacity as the sum of the input line capacities because each terminal may have data to transmit when its time slot is serviced. When a terminal has no data to transmit, the output line transmits dummy fill characters during that terminal's time slot.

The big disadvantage of Time Division Multiplexing (TDM) is that when a terminal has no traffic, then its time slot is wasted. It is not possible to simply fill in the unused time slot with data from another terminal because both sender and receiver are synchronized as to when a specific terminal's time slot occurs. No mechanism exists to key the receiver that the agreed upon position sequencing of terminals has changed without resynching the two ends. If each terminal has traffic only a small fraction of the time, then TDM makes inefficient use of the output line capacity [5:121].

3.3.1.2 Terminal Concentration - When the actual traffic is far below the potential traffic, most of the time slots on the output line are wasted. Consequently, it is often possible to use an output line whose transmission capacity is less than the sum of the input line capacities. This arrangement is called Concentration. The usual approach is

to only transmit actual data and not dummy fill characters. However, this strategy introduces two new problems.

The first problem is keying the receiver as to which characters came from which input line [5:122]. To solve this problem, the string of characters from each terminal is arranged into a message to which is appended a message header prior to its release to the output line. This message header contains (among other fields) the terminal identification of the sender.

The other problem is tied to the smaller line capacity of the output line. If each terminal suddenly starts outputting data at its maximum rate, inadequate output line capacity exists to handle the deluge and some data may be lost. For this reason, concentrators are always provided with extra data buffers in order to survive short data surges [5:122]. The more memory (larger the buffers) that a concentrator has, the more it costs, but the more likely it is to survive the short data surges. Choosing the appropriate parameters for output line bandwidth and concentrator memory size involves trade-offs. If either is too small, data may be lost. If either is too large, then the entire arrangement may be unnecessarily expensive. Furthermore, the optimum choices depend upon the traffic statistics, which are not always known at system design time [5:122].

The FEP system will provide Terminal Concentration at the LTM node by merging the asynchronous serial inputs from the seven user terminals into a single data flow to the LLM node. Although traffic statistics are not known at this time, the DMA transmission rate is known to be as high as 125000 words per second up to lengths of 50 feet [18:2.1].

3.3.1.3 Packet Assembly/Disassembly - Packet Assembly/Disassembly (PAD) was renamed Command Completion Sensing PAD and is deferred treatment until the Transport Layer Protocol discussion (paragraph 3.3.4.6).

3.3.1.4 Error Control - Most Error Control at the Physical level is realized in hardware. Since plans do not exist to add hardware components, Error Control will be deferred treatment until the Data Link Layer discussion (ref para 3.3.2.1).

3.3.2 Data Link (Layer 2) -

Typical issues at this level include Frame Control [5:137], Buffering and Flow Control [5:143], Sequence Numbering [5:146], and Error Control [5:164]. The only node-to-node path along which data transverses a physical data link is at the LLM/VLM interface. Therefore, the data link protocol will only apply to that interface. The primary network effect characterizing this level is the electrical noise in the physical medium. Resolution of

several Data Link issues normally requires the construction of a Data Link Frame Header (DLFH). Figure 3-3 contains the format for the FEP DLFH.

3.3.2.1 Error Control - Error Control, deferred from the Physical level (para 3.3.3.4), is provided by placing a checksum within the Data Link Frame Header (DLFH). The receiving node verifies this checksum with an independent one which it calculates itself. If the checksums verify, then the receive node "piggy-backs" an acknowledgement (ACK) back to the sending node in the DLFH of the next reverse direction message frame. If reverse traffic does not occur prior to a timeout period, then the ACK is sent as a new packet. If the checksums do not verify, then an immediate NAK is sent to the sender who responds by retransmitting the frame.

3.3.2.2 Frame Control - An ACK/NAK timeout is used to ensure positive receipt of the message frame. Automatic retransmission occurs for the frame currently outstanding if the ACK/NAK timer expires prior to some acknowledgement of receipt from the receiver.

3.3.2.3 Buffer and Flow Control - The DMA channel could become a bottleneck in the system. Therefore, adequate Buffering and Flow Control mechanisms must be used. This is true for both ends, but is most critical at the LLM (LSI)

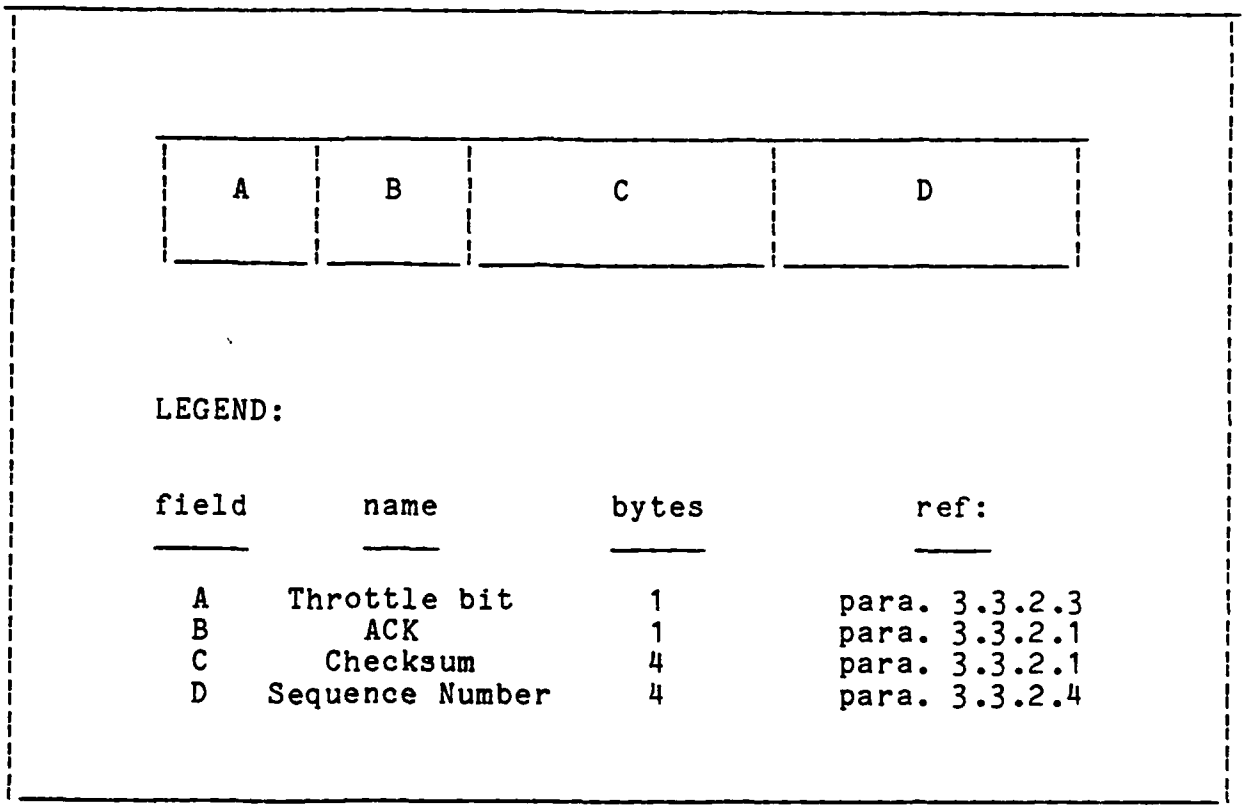


Figure 3-3 Data Link Frame Header (DLFH)

end because file transfers from the VAX could conceivably saturate the LSI node. To provide Flow Control, a "throttle bit" is provided within the DLFH. By this method, the LLM could ACK the last frame (and there-by inhibit a retransmission) but still alert the VLM to stop transmitting until the ACK was repeated with the "throttle bit" turned off.

3.3.2.4 Sequence Numbering - At times, the situation may occur in which the receiver acknowledges receipt of a message, but an electrically noisy medium damages (or loses) that acknowledgement (ACK). According to para 3.3.2.2, the currently outstanding frame would be retransmitted when its ACK/NAK timer expired. The receiver would then receive a duplicate frame of the one it had just acknowledged. To preclude the receiver from again processing this message request a sequence number is included in the DLFH (fig 3-3). The transmitter increments this number for each newly created message frame. The receiver ACKs each frame it receives but only processes messages containing new sequence numbers.

3.3.3 Network (Layer 3) -

Typical issues at this level include Error and Sequence Control [5:190], Routing [5:196-214], Buffering and Congestion Control [5:215-225], and Accounting. The network

is passing data "packets" between logical nodes at this level.

3.3.3.1 Error Control - Error Control, at this level, only applies to delayed packets because most data transfers between logical nodes occur within the same machine (VAX or LSI). This packet passing consists of passed parameters and common buffers between called subroutines. Therefore, in the absence of an electrically noisy medium, traditional issues of lost, damaged, or duplicate packets are not a problem at this level.

3.3.3.2 Sequence Control - Sequence Numbering is deferred to the Transport layer (para 3.3.4.5).

3.3.3.3 Buffering and Congestion Control - Congestion Control is provided by implementing interface Buffers between each pair of logically connected nodes. Four pointers (LOW, PUT, GET, HIGH) are defined for each buffer. LOW and HIGH define the physical limits of the buffer in memory and never change. The Sending node writes to the buffer and adjusts the PUT pointer. The receiving node reads the buffer and adjusts the GET pointer. If sufficient room does not exist within the buffer for the packet in hand, then the sending node throttles itself and checks again later. This ensures that no logical node is sent data for which it does not have sufficient buffer space

to contain it.

Although circular buffering was preferred, system limitations required that flat buffering be used in the final implementation. This limitation revolved around the DMA hardware which incremented an addressing register [18:4-9] to point to the next word for data transfer. This circuitry could not be programmed to account for buffer "wrap-around" situations. Thus, all data words constituting a single message were constrained to be stored in contiguous memory locations.

3.3.3.4 Routing - Since a unique path exists between each user (LSI) and process (VAX) pair, alternate routing is not a consideration. Once the LSI's terminal ID (TID) and the VAX's process ID (PID) have been established, the virtual circuit (parts, of which, are shared by other virtual circuits) will be known and remain static throughout the terminal session.

3.3.3.5 Accounting - Accounting statistics are maintained for queueing and computer performance evaluation studies. Statistics are updated each time that a change in queue status occurs. These updated statistics are spooled out to disk for off-line data reduction and analysis.

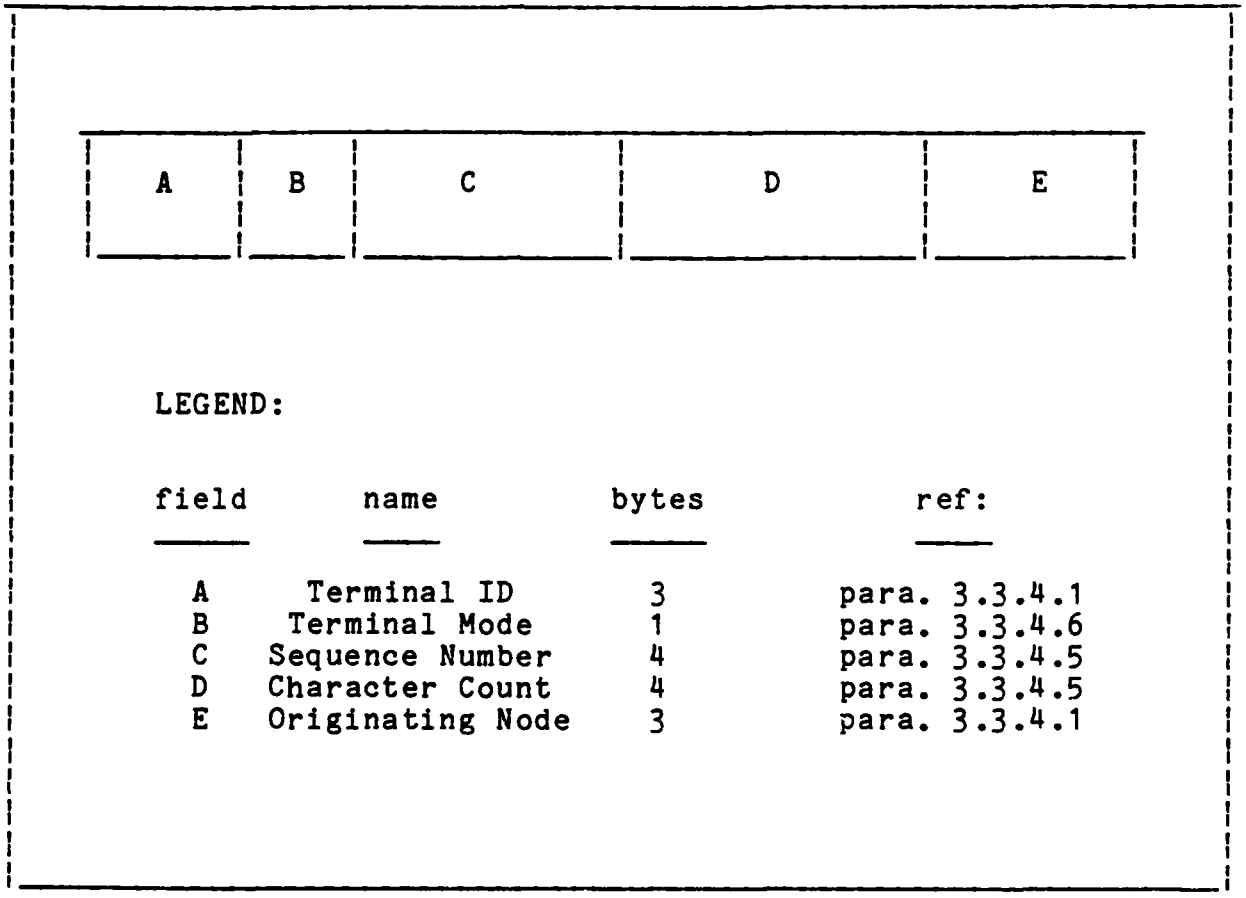
3.3.4 Transport (Layer 4) -

Issues at this level include Addressing and Connection [5:325-338], Flow Control [5:338-343], Process Multiplexing [5:343-345], Error Control [6], Sequencing and Segmentation [6], and Command Completion Sensing PAD, which was deferred from the Physical Layer (ref para 3.3.1.3). Similar to the Data Link Frame Header (ref para 3.3.2), a Transport Header (ref Fig 3-4) is required to resolve issues at this level.

3.3.4.1 Address and Connection - Connection/Termination at the Transport level consists of the "LOGON" and "LOGOFF" requests and VAX responses. Within each Transport Header (ref Fig 3-4) is a field set aside for Terminal ID. The Terminal ID (one of 8 possible values representing one of the 8 peripherals attached to the LSI) also constitutes the Circuit ID in a one-to-one mapping.

The Transport Header (ref Fig 3-4) also contains a Node Identification field. This Node ID indicates from which node queue the message originated. The primary purpose of the Node Id is to tag the Accounting data so that queueing statistics can be calculated.

3.3.4.2 Flow Control - Flow Control and Buffering techniques are the same at this level as those employed at the Network level (ref para 3.3.3.3). Maximum buffer size



LEGEND:

field	name	bytes	ref:
A	Terminal ID	3	para. 3.3.4.1
B	Terminal Mode	1	para. 3.3.4.6
C	Sequence Number	4	para. 3.3.4.5
D	Character Count	4	para. 3.3.4.5
E	Originating Node	3	para. 3.3.4.1

Figure 3-4 Transport Header

will be initially set at 2000 characters and adjusted, if required, during the implementation and testing stages when empirical data becomes available.

3.3.4.3 Process Multiplexing - Upward Multiplexing is used within the VPM node to transfer the terminal requests to the proper VAX/VMS process. In this scheme, multiple Transport connections (VT-100 terminals) all use the same virtual circuit (DMA) to the host.

3.3.4.4 Error Control - Error Control is not required at this level due to the method of internally passing the data between nodes (ref para 3.3.3.1).

3.3.4.5 Sequencing and Segmentation - Sequencing is normally required at this level in case a large message must be fragmented into smaller chunks to transit the network boundaries.

At the Transport level, the data entity is the Message. The Message size is variable so that bandwidth is not wasted. This requires a field in the Message Header to specify the Message size. The maximum Message size is a function of the maximum VAX data transfer rate. If this number is large (i.e. - 25 terminal lines X 80 chars/line = 2000 chars), then the message may have to be Segmented with Transport Header Sequencing information to identify the parts (packets) of the Message while it is moving about

within the network. The receiving end collects all the message parts (packets) and reconstructs the original message in its proper order.

This feature is only needed for large file transfers. The only large file transfer within the FEP system consists of the host sending the terminal a large display file. Therefore, Message Segmentation would normally only be required at the VAX node (VPM) while Message Reconstruction would normally only be required at the LSI (LTM) end.

However, even this limited application of segmentation is not really required. Since there can be, at most, one single outstanding frame at any given time (ref para 3.3.2.2), there is no possibility that frames will be received out of order. Also, there is no special action that the LSI FEP software must perform upon receipt of process-to-screen display traffic. It is transparent to the LSI FEP software whether the characters forwarded to the terminal arrived as part of several independent messages or as segmented frames of one large message. Therefore, segmentation and reconstruction would introduce unrequired complexity into the system.

However, Transport Header sequencing (ref Fig 3-4), mainly used at this level for segmentation, will be retained as part of the "Support for LCN Study" (ref para 2.2.2.3)

requirement. This Message Sequence Number will be used to track the movement of the message through the network nodes and queues.

3.3.4.6 Command Completion Sensing PAD - One of the LSI (LTM) responsibilities is to assemble a complete interactive request and ship it to the VAX. Command Completion Sensing is the software recognition that an input line (assembled from the terminal) is complete and ready for VAX processing. Normally, a carriage return signifies the point at which the request is complete (LINE mode). However, certain processes expect and respond to individual keystroke commands (CHARACTER or WORD mode) [7:3.4]. For those processes, the LTM node must be "clued-in" that the process responds to single character input. Therefore, a Mode field is included in the Transport Header (ref Fig 3-4) where-by the VPM node will inform the LTM node of the current "mode" of operation.

3.3.5 Session (Layer 5) -

At this level, a Network Status and Control function exists to allow the System Manager to initialize, display status, and terminate the network. System level alerts and warnings are generated and sent to the respective system consoles (LSI SOC or VAX SOC).

3.3.6 Presentation (Layer 6) -

Functions at this level typically provide the user with certain useful, but not always essential, services. Among these services are cryptographic transformations, text compression, terminal handling, and file transfer [5:386]. These functions are either a) not required for the FEP system, or b) addressed at a lower protocol level. Therefore, no Presentation issues were designed in this investigation.

3.3.7 Application (Layer 7) -

The only Application function identified within the system is the Accounting (and disk recording) of the queueing statistics.

3.4 Summary -

This chapter described the FEP system from a network perspective. Logical and physical nodes were defined and identified. The ISO protocol model was used to explain the services and requirements at each level of the network hierarchy. Techniques for data movement and control within the network were discussed. Certain traditional protocol

requirements were eliminated and the rationale used to reach these decisions was presented.

CHAPTER 4

SOFTWARE DESIGN AND IMPLEMENTATION

This chapter describes the structural design and implementation details of the FEP software which was expected to execute on the LSI-11/23. It begins with a discussion of the Software Capabilities and Limitations which influenced and bounded the implementation effort. Next, the Software Conventions used within the program are discussed. Then, an Overview of the Software Structure provides a skeletal outline of the program processes. Finally, the Synopsis of Program Modules describes the processing steps within each major software module.

Supplementary material can be found in Appendix B (LSI FEP Structure Charts), Appendix C (LSI FEP Data Dictionary), Appendix D (LSI FEP Source Code Listings), Appendix E (LSI FEP Memory Load Map), Appendix F (LSI FEP User's Guide), and Appendix G (LSI FEP Programmer's Guide).

4.1 Software Capabilities And Limitations -

The 'C' language was selected for LSI FEP implementation because of its availability and power of expression beyond PASCAL. The version of 'C' run on the LSI-11/23 [21; 23] implemented most of the language Capabilities [22]. Those capabilities and limitations which are of particular importance to this thesis effort are discussed in the following paragraphs.

4.1.1 Structured Constructs -

'C' provides the fundamental flow-control constructions required for well-structured programs: decision making (IF - ELSE); looping with termination at the top (WHILE, FOR) or at the bottom (DO); and selecting one of a set of possible cases (SWITCH) [22:3]. Although non-structured GOTO and LABEL constructs are provided within the 'C' language, neither was used within this thesis implementation in order to preserve the top-down execution flow.

4.1.2 Data Structures -

The fundamental 'C' data structures are characters, integers, and floating point numbers. In addition, there is a hierarchy of derived data types created with pointers, arrays, structures, unions, and functions. 'C' provides pointers and the ability to do address arithmetic. The arguments to functions are passed by copying the value of the argument. Therefore, it is impossible for the called function to change the actual argument in the caller. When it is desirable to achieve "call by reference", a pointer may be passed explicitly, and the function may change the object to which the pointer points [22:3].

4.1.3 Global vs. Automatic Variables -

Any function may be called recursively, and its local variables are "automatic" or created anew with each invocation. When the function terminates, its automatic variables are destroyed. Other than locally defined within the function, its variables may be external (but known only within a single source file) or completely global. Both of the latter types remain defined throughout program execution and can be referenced by all program functions [22:3].

4.1.4 Variable Name Lengths -

A program written in 'C' must be compiled by the 'C'

compiler --- producing a macro file which must then be assembled using the RT-11 Macro Assembler. Therefore, the 'C' source program must comply with the syntax rules of both languages in order to produce errorless object code. The next paragraph indicates a few areas where conflicts exist between the syntax rules of the compiler and those of the assembler.

Names are made up of letters and digits; the first character must be a letter in 'C' [22:33], but may be a number for the RT-11 Macro Assembler [15:3-6]. The underscore "_" counts as a letter in 'C' [22:33], but is rejected as an invalid character by the RT-11 Macro Assembler [15:3-6]. Only the first 8 characters of an internal name are significant in 'C', although more may be used [22:33]. However, the RT-11 Macro Assembler only recognizes the first 6 characters as unique and significant [15:3-6].

These inconsistencies forced a more cryptic naming of certain functions and variables than would have been desired.

4.1.5 Limited Symbolic Definition Capability -

Although the vendor documentation [21] does not reference it, an upper bound does exist for the number of user defined symbolic constants that Telecon 'C' will

support for each program [23:6]. These globally defined constants are desirable from a software engineering aspect due to the resulting ease of software maintenance. Although the exact threshold was never calculated, "LSIFEP.C" exceeded it several times during the development effort. Each such instance was resolved by eliminating some desirable symbolic constant and replacing it with its hard-coded value wherever it was referenced.

4.2 Module Communication Conventions -

Very few conventions were designed into the program in order to simplify its maintenance. A small set of global variables were declared in order to minimize the intermodule data-flow complexity. Most intermodule communication takes place via character arrays, structure tables, and indexes into both.

4.2.1 Character Arrays -

Three character arrays exist. "InChar" is an 1800 element array divided into 9 parts (8 terminals ports + 1 DMA port), each containing storage for 200 characters. This array accepts characters via interrupt service routine processing of terminal keyboard input characters. "OutChar" is an identically configured array which contains output

characters destined for output to one of the 8 terminal display screens. "NodeChar" is a 4000 element array divided into 2 parts (terminal-to-DMA and DMA-to-terminal traffic) containing storage for 2000 characters each. This array serves as a queueing buffer for complete message transfers between network nodes.

4.2.2 Structure Tables -

Three structure tables exist. The Port Status Table ("PST") contains buffer indexes, I/O port addresses, and other information desribing the status of each of 9 ports. The Node Buffer Table ("NBT") contains buffer indexes and the node identification information required for moving message traffic between the two LSI FEP nodes. The Transport Header Table ("THT") contains the Message Transport Header skeleton which is prefixed to each message prior to movement of that message to one of the node queues.

4.3 Overview Of The Software Structure -

Utilizing the RT11XM extended memory features, the LSIFEX.SAV executable image was created by specifying to the RT-11 Linker the order and relative locations desired for the seven object modules which form the LSI FEP software. Likewise, the low memory features of the RT11SJ monitor were

used to create the "LSIFEP.SAV" program image.

The source code programs which generate these seven modules are classified according to placement within memory and are discussed in the following sections. Program names ending with ".C" are 'C' source programs while those ending with ".MAC" are macro assembly language programs. A Load Map of "LSIFEX.SAV" is contained in Appendix E. Figure 4-1 contains a graphic representation of central memory placement of the software modules within "LSIFEX.SAV".

4.3.1 Low Memory Software -

Software placed within the confines of low memory included those functions and data areas which were constrained (ref para 2.3.7) to reside there and those data areas (primarily NBUFF.MAC) which served as filler to ensure that the PAR1 restrictions (ref para 2.3.7) were met.

4.3.1.1 LFEPIO.C - The LSI FEP Input/Output 'C' module is a derivation of the Standard I/O package (STDIO.H) included with the 'C' compiler [22:143 and 21:8]. Routines non-essential to the LSI FEP software and/or not supported by the Telecon 'C' compiler were removed in order to save memory.

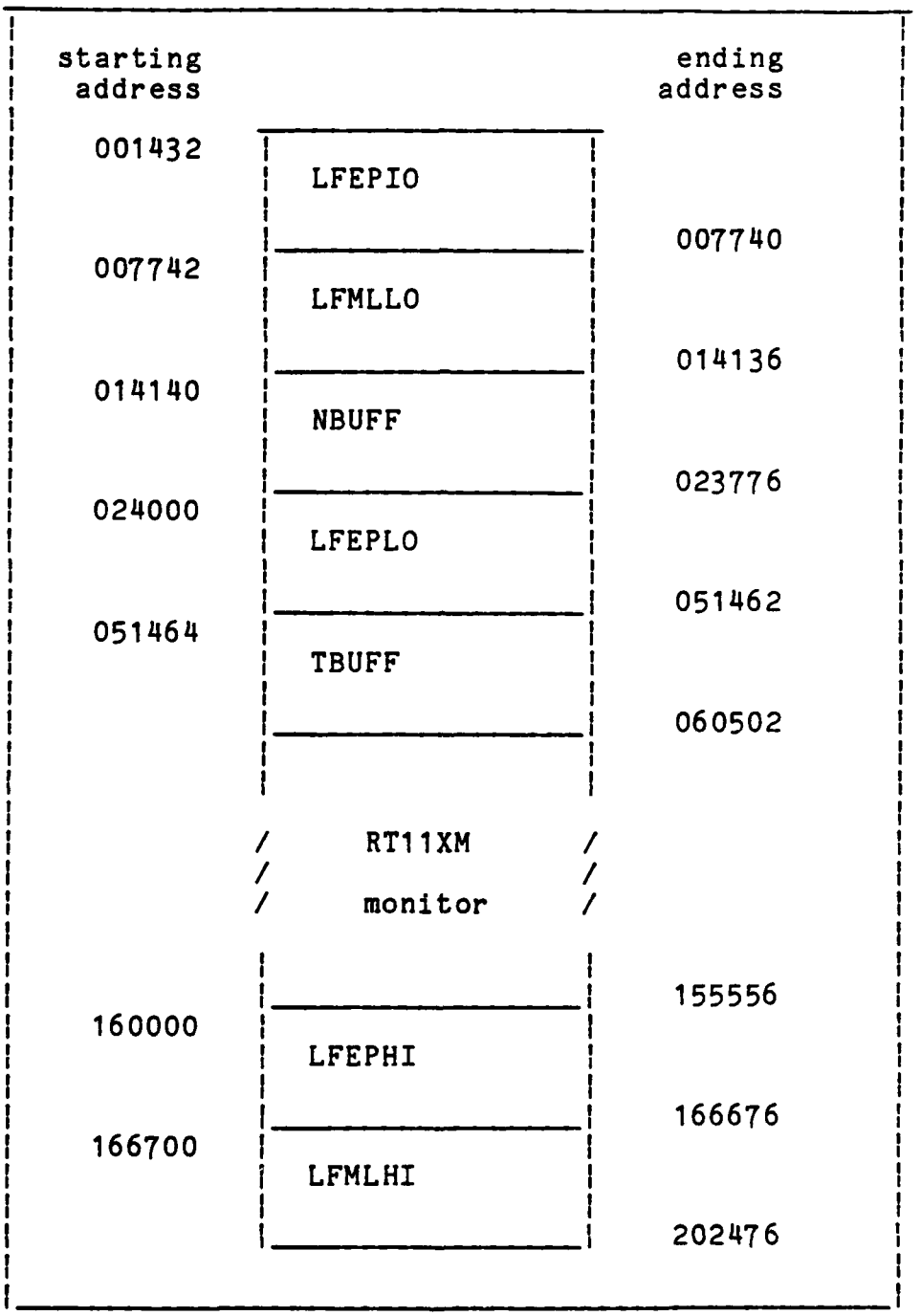


Fig. 4-1 LSIFEX Memory Layout

4.3.1.2 LFMLLO.MAC - The LSI FEP Macro Library (Low Memory) assembly module is a derivation of the Telecon 'C' Runtime Support Library "CLIB11.MAC" [21] as augmented by previous classroom (EE6.90 - Real Time Programming Laboratory) upgrades. Again, non-essential functions were removed. The remaining program was further divided into a portion (LFMLLO.MAC) that was constrained (ref para 2.3.7) to reside in low memory and a portion (LFMLHI.MAC) that could reside in extended memory.

4.3.1.3 NBUFF.MAC - The Node Buffer macro program contains the data item definitions which define the size and start address of the character array "NodeChar".

4.3.1.4 LFEPLO.C - The LSI FEP (Low Memory) 'C' program contains the main functions which implement the LSI FEP processing. Program initialization, interrupt servicing, background processing, and program termination are contained there.

4.3.1.5 TBUFF.MAC - The Terminal Buffer macro program contains the data item definitions which define the sizes and starting addresses of the terminal character arrays "InChar" and "OutChar".

4.3.2 Extended Memory Software -

Two software programs were placed into the extended

memory portion of the LSI-11/23 physical address space. These programs contained no functions which were called by interrupt processing routines. Therefore, the PAR1 restriction (ref par 2.3.7) did not apply to them.

4.3.2.1 LFEPHI.C - The LSI FEP (High Memory) 'C' program contains routines called during synchronous processing events. Typically, these routines provide some low priority processing such as displaying some aspect of the system status upon the SOC terminal screen or recording the accounting statistics.

4.3.2.2 LFMLHI.MAC - The LSI FEP Macro Library (High Memory) assembly module contains those functions (ref para 4.3.1.2) which could be removed to extended memory. Typically, these functions provided communication with the floppy disk files.

4.4 Synopsis Of Program Modules -

Program modules reside either in low memory or in extended memory.

4.4.1 Low Memory Modules -

The following modules perform the main functions of the LSI FEP software.

4.4.1.1 Main - This module (module 0) is the top segment to which control is transferred by the 'C' Shell upon program initiation. This module calls "InitSystem" to initialize the database and activate interrupts. It then calls "PerfNormalActivities" to perform the synchronous tasks (move data between nodes, output characters to the terminal screens, etc.). When the SOC operator aborts the system (ref. paras 4.4.1.5 and 4.4.1.15), then control is returned to this module which then calls "TermSystem" to deactivate interrupts and close files. The module finishes its processing by an exit to the RT11XM monitor.

4.4.1.2 InitSystem - This module (module 1) controls the database initialization. It begins processing by opening the LSIFEP.DAT accounting file. It then sets up the four pointers (ref para 3.3.3.3) into the terminal buffers "InChar" and "OutChar" for each of the nine entries in the Port Status Table (PST). It then sets up the four pointers into the "NodeChar" buffer for each of the two entries in the Node Buffer Table (NBT). It then moves the address of each interrupt service routine into the corresponding entry of the PST for easy retrieval by the "InitInterrupts" routine. It then calls "InitPST" to complete the PST initialization. It then calls "InitInterrupts" for each entry in the PST to initialize the interrupt for that port.

4.4.1.3 InitPST - This module (module 1.1) begins processing by coding each PST entry with a unique terminal identification (TID). It then fetches and moves to the PST the addresses of the four port I/O interface registers: Receiver Control and Status Register (RCSR), Receiver Buffer Register (RBUF), Transmitter Control and Status Register (XCSR), and Transmitter Buffer Register (XBUF) [20:221-223] for each of the nine entries in the PST. It also moves the address of each interrupt vector [20:508-509] to the nine entries of the PST. It completes processing by initializing the terminal mode (ref para 3.3.4.6) to "LINE" mode.

4.4.1.4 InitInterrupts - This module (module 1.2) activates the interrupts for each of the nine ports defined in the PST. It begins processing by setting up a Processor Status Word (PSW) mask [20:176-177; 207-209]. It then saves (in the PST) the current contents of the port's interrupt vector and interrupt PSW. It then resets these two locations to the address of the interrupt service routine (fetched in "InitSystem") and the new PSW mask just generated. It then turns on the Interrupt Enable bit (bit 6) of the corresponding RCSR [20:221].

4.4.1.5 SOCInterruptServiceRoutine - This module (module 1.2.1) executes when control is passed to it by the RT11XM monitor in response to a console keyboard action at the System Operator Console (SOC). It begins processing by

calling "entint" to save the register values of the interrupted program. It then copies the input character to the input queue "InChar" and echoes the character to the console screen. If the character was a carriage return, the module echoes a line feed character to the screen. If it was a Control-C (^C), special termination processing occurs. If it was a "delete" character, special processing occurs. If it was neither, the InChar "put" index is incremented for the next character.

The ^C input signals the SOC operator's intention to abort the LSI FEP system. This input results in the module setting the "AbortFlag" boolean variable to "YES". This flag is checked in module "PerfNormalActivities".

A "delete" character requires special treatment because, instead of adding characters to "InChar", this action results in withdrawing characters. Processing consists of verifying that the "delete" character was not the first character typed. This ensures that at least one character already exists in the buffer and can be deleted. If such a character exists, then the "InChar" "put" index is decremented so that the next input character will over-write the character intended for deletion. Then, to clean-up the display terminal, a series of "backspace" and "space" characters are output to blank out the deleted character and reposition the screen cursor.

Processing terminates during a call to "retint" which restores the register contents of the interrupted program and executes the "RTI" (return from interrupt) assembly language instruction.

4.4.1.6 T1InterruptServiceRoutine - This module's (module 1.2.2) processing is identical to that of module 1.2.1 except that it performs no special processing for the ^C input and it services character input from terminal 1.

4.4.1.7 T2InterruptServiceRoutine - This module's (module 1.2.3) processing is identical to that of module 1.2.2 except that it services characters input from terminal 2.

4.4.1.8 T3InterruptServiceRoutine - This module's (module 1.2.4) processing is identical to that of module 1.2.2 except that it services characters input from terminal 3.

4.4.1.9 T4InterruptServiceRoutine - This module's (module 1.2.5) processing is identical to that of module 1.2.2 except that it services characters input from terminal 4.

4.4.1.10 T5InterruptServiceRoutine - This module's (module 1.2.6) processing is identical to that of module 1.2.2 except that it services characters input from terminal 5.

4.4.1.11 T6InterruptServiceRoutine - This module's (module 1.2.7) processing is identical to that of module 1.2.2 except that it services characters input from terminal 6.

4.4.1.12 T7InterruptServiceRoutine - This module's (module 1.2.8) processing is identical to that of module 1.2.2 except that it services characters input from terminal 7.

4.4.1.13 DMAInterruptServiceRoutine - This module (module 1.2.9) services all DMA interrupts. It begins processing by determining the reason for the interrupt. If non-existant memory was accessed, it sends an error alert to the SOC. If the host has raised an input request, this module calls "SetUpForInputDMA" to process the request. If none of the above reasons, it bases further processing upon the last reported status of the "DMABusyFlag".

If a word mode input was expected, the word is fetched and inspected. This word represents the host's word count for an ensuing block mode transfer of data across the DMA channel. If sufficient buffer space exists in "NodeChar" to hold a message of this many characters (twice the word count), the DMA interface is programmed [18:chapter 4] to expect a block mode DMA input from the host. If buffer space does not exist, an error alert is issued to the SOC screen.

If a block mode input was expected, then this interrupt notifies the LSI that the block transfer has completed. The module adjusts the NBT pointers into "NodeChar" and resets the "DMABusyFlag" to input word expected.

If a word mode output was in progress, then this interrupt signals the host response to the block mode output request. If the host is prepared to accept the data block, the DMA interface is programmed for block mode output and the "DMABusyFlag" is set to block mode output in progress.

If a block mode output was in progress, then this interrupt signals the completion of the output transfer. The "DMABusyFlag" is reset to word mode input expected and the DMA interface is programmed accordingly.

4.4.1.14 SetUpForInputDMA - This module (module 1.2.9.1) begins processing by setting the "DMABusyFlag" to word mode input expected. It then programs the DMA interface accordingly.

4.4.1.15 PerfNormalActivities - This module (module 2) begins its processing by displaying upon the SOC terminal the time at which the LSI FEP system was activated. It then enters the large system idle-loop (ref. section 2.3.2) which encompasses all synchronous processing tasks. The software will continue to idle within this loop as long as the data item "AbortFlag" remains equal to the boolean constant "NO" (defined as 0). When the SOC operator keys in a control-C (^C), then "AbortFlag" is set to the boolean "YES" (defined as 1). When the bottom of the loop is reached, the status of "AbortFlag" is checked. If it equals

"YES", then the statement following the loop (return to "Main") is executed.

Within the loop, three main synchronous tasks are done. For each terminal, if new characters have been deposited in the terminal input buffer by the interrupt service routines, then "SrvInputQueue" is called to service that input queue. Likewise, if output characters have been deposited in one of the output queues, then "SrvOutputQueue" is called to service that output queue. If a message exists in one of the node queues, then "SrvNodeQueue" is called to service that message.

If no work exists when a particular queue is checked, then the "put" and "get" pointers are reset to their "low" value. This step is required because a previous design decision (ref para 3.3.3.3) disallowed circular buffering. By synchronously resetting empty buffer pointers to their initialized values (ref para 4.4.2), the software ensures that the full buffer capacity is available for the next data entry.

4.4.1.16 SrvInputQueue - This module (module 2.1) scans the input character buffer "InChar". If the character currently being scanned is a carriage return or if the terminal is in the "CHARACTER" mode, then a complete user request is present and can be assembled for DMA transfer to

the host. Otherwise, the next character in the buffer is scanned and similar testing performed until all the characters in the buffer have been scanned.

If a complete request exists in "InChar", then a check is made as to whether the SOC buffer is being scanned. If yes, then module "EvalSOCInput" is called to determine if DMA transfer is required. After this test, and if DMA transfer is required, then a character count is calculated for inclusion in the Transport Header (ref para 3.3.4.5). Since the DMA interface requires full word (two character bytes) transfers [18:4.9], an odd number value for the message size is incremented to make it even.

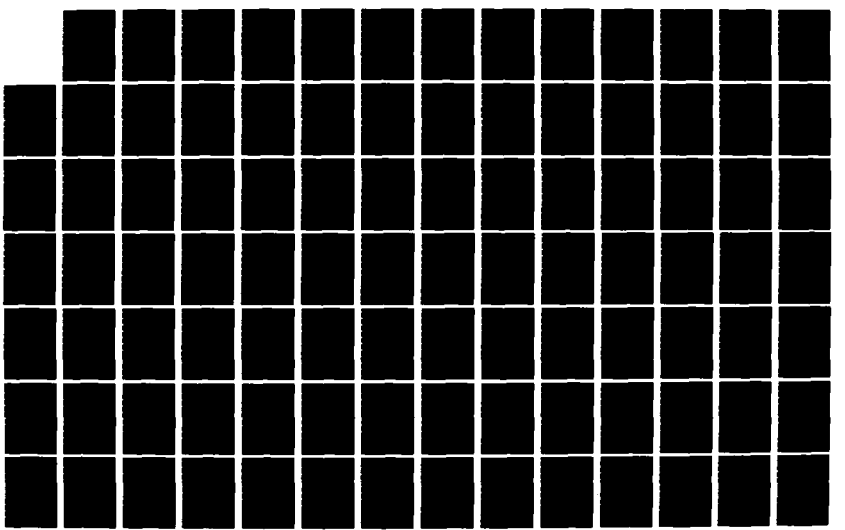
If buffer space exists to hold a message of this character count, module "MoveMsgtoNodeTTxDMA" is called. Otherwise, an error alert is issued to the SOC screen.

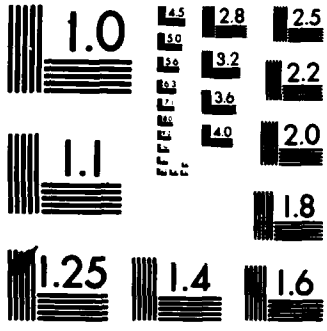
4.4.1.17 EvalSOCInput - This module (module 2.1.1) determines if a SOC system status request has been issued. If so, then this request will be processed locally within the LSI and not forwarded to the host. If not, then the "goDMAFlag" is set to "YES" for "SrvInputQueue" processing.

If the SOC request is to display the PST, then "DispPST" is called. If the SOC request is to display the NBT, then "DispNBT" is called. If the SOC request is to display the current system time, then "DispTime" is called.

AD-A138 152 DEVELOPMENT OF A COMMUNICATIONS FRONT END PROCESSOR (FEP) FOR THE VAX-11/.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. A F MASTY

UNCLASSIFIED DEC 83 AFIT/GCS/EE/83D-13 F/G 17/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

4.4.1.18 MoveMsgtoNodeTTxDMA - This module (module 2.1.2) is called by "SrvInputQueue" when a DMA output transfer is required. This module calls module "BldTransportHeader" to construct the Transport Header (ref Fig 3-4). It then moves the Transport Header characters into "NodeChar" followed by the "InChar" terminal request characters. If the number of characters in the terminal message is odd, then a harmless line-feed character is added to "NodeChar" to pad the message into an even word block. Module "GatherStats" is then called to record the accounting data of this message's entry into a queue.

4.4.1.19 BldTransportHeader - This module (module 2.1.2.1) constructs the Transport Header as defined in Figure 3-4. Processing consists of the movement to the Transport Header Table (THT) skeleton of either single character bytes or strings of bytes (via calls to the "strcopy" module). Also, integer to ASCII character conversion of numeric quantities is accomplished by call to subroutine "intascii".

4.4.1.20 SrvOutputQueue - This module (module 2.2) moves character data from "OutChar" to the Transmitter Buffer Register (XBUF) for display upon the user terminal screen. If the Transmitter Control and Status Register (XCSR) transmit ready bit (bit 7) is set = 1, then a new character can be moved to the XBUF. Otherwise, this module returns to

"PerfNormalActivities" to continue other processing.

4.4.1.21 SrvNodeQueue - This module (module 2.3) controls the movement of messages between LSI FEP network nodes. Two nodes exist. The first node (entry 0 in NBT) is labeled TTxDMA and is used to hold message traffic input from the terminal which is intended for output along the DMA interface. The other node (entry 1 in NBT) is labeled DMATTx and is used to hold the DMA input message traffic which is destined for output to the terminal screen.

Processing begins by scanning the Transport Header (fig 3-4) to determine message length and terminal identifier (TID). Subroutine "strcompare" is called to compare the message TID with those recognized by the system and stored as entries in the PST. If no match is found, an error alert is issued to the SOC terminal screen and the node is flushed of all message data by re-initializing the "NodeChar" indexes.

If a match is found, then the ASCII message character count is fetched from the Transport Header and converted to integer via a call to "asciiint". If the node being scanned is TTxDMA, then module "TTxtoDMAOutput" is called. Otherwise, "DMAtoTTxOutput" is called.

4.4.1.22 TTxtoDMAOutput - This module (module 2.3.1) activates the DMA output request to the host computer. If the "DMABusyFlag" status indicates word mode input expected and the DMA Control and Status Register (DMACSR) reports the DMA idle, then the DMA interface is programmed for word mode output and the "DMABusyFlag" is set to word mode output in progress. The word that is output is the character count of the message which would be sent to the host in block mode. Subroutine "GatherStats" is called to record the release of message data from the TTxDMA queue.

If either condition is not met, an error alert is issued to the SOC terminal screen.

4.4.1.23 DMAtoTTxOutput - This module (module 2.3.2) controls the movement of message data (received from the DMA interface) from the DMATTx buffer to the appropriate "OutChar" buffer. First, "GatherStats" is called to record the release of message data from the DMATTx queue. Next, the Terminal Mode field in the Transport Header (Fig 3-4) is moved to the corresponding PST entry. Finally, each message character is copied from the "NodeChar" buffer to the appropriate "OutChar" buffer and pointers adjusted accordingly.

4.4.1.24 TermSystem - This module (module 3) is executed once after "AbortFlag" has been set to "YES" in

"SOCInterruptServiceRoutine" and, in response, the large loop in "PerfNormalActivities" terminates and returns control to the top segment "Main".

"TermSystem" restores all interrupt vectors and Processor Status Words (PSW) (ref para 4.4.1.4) to their original contents. A message is then output to the SOC terminal indicating the time at which the "abort" command was processed and the duration of the LSI FEP processing.

File "LSIFEP.DAT" is then closed and the software exits to the RT-11 monitor.

4.4.2 Extended Memory Modules -

The following paragraphs describe the 'C' modules which have been mapped to extended memory. These modules typically provide services to the synchronous processing modules described earlier (ref para 4.4.1). Since these services possess no hierarchical relationships to each other, the module ordering conventions were arbitrarily assigned as the need for a new service became known. The module numbers represent the order in which the modules appear in the program section 'LFEPHI.C'. To preclude confusing these module numbers with those of 'LFEPLO.C', each extended memory module number will contain an 'X' prefix.

4.4.2.1 DispPST - This module (module X.1) displays the status of selected fields of the Port Status Table (PST) upon the SOC terminal screen. The format of this display is contained in Figure F-4 of Appendix F.

4.4.2.2 DispNBT - This module (module X.2) displays the status of selected fields of the Node Buffer Table (NBT) upon the SOC terminal screen. The format of this display is contained in Figure F-3 of Appendix F.

4.4.2.3 GetCurrentTime - This module (module X.3) calls an assembly language routine, 'gtime', which fetches the current system time and returns it in a two word parameter table. This module then decodes these two words, converts the numbers to ASCII characters and produces a displayable time in the format of HH:MM:SS:TT where HH = hours, MM = minutes, SS = seconds, TT = ticks (60 ticks per second).

Four global data items ('StartHr', 'StartMin', 'StartSec', 'StartTic') are defined to contain the initial start-up time of the LSI FEP system. These start-up values are subtracted from the time calculated at system termination to provide the elapsed time for the LSI FEP operation. These data items are initialized with the current system time during the initial calling of 'GetCurrentTime'.

4.4.2.4 DispTime - This module (module X.4) calls 'GetCurrentTime' (module X.3) and then displays the formatted time upon the SOC terminal screen.

4.4.2.5 CalcElapsedTime - This module (module X.5) is called once - at LSI FEP system termination. This module calls 'GetCurrentTime' (module X.3) to fetch the current system time. It then subtracts this time from the four start-up values and provides an elapsed time.

4.4.2.6 DispElapsedTime - This module (module X.6) calls 'CalcElapsedTime' (module X.5) to calculate the elapsed time. It takes the elapsed time and converts the integer values to ASCII characters and displays the result upon the SOC terminal in the form HH:MM:SS:TT.

4.4.2.7 GatherStats - This module (module X.7) records the accounting data for LCN queueing study (ref para 2.2.3.3). Calling parameters to this subroutine are a pointer to a character string which forms a Transport Header (ref Fig 3-4) and a reason code specifying which action (1 = queue entry, 2 = queue exit) is being recorded. Processing begins with a call to 'GetCurrentTime' (module X.3) to fetch the current time. The time, action code, and Transport header are then written to the disk file 'LSIFEP.DAT' in the format specified in Figure F-5.

4.5 Summary -

This chapter described the software design and implementation details. It began by discussing the capabilities and limitations of the Telecon 'C' language. Next, programmer conventions for inter-module software communication were outlined. The chapter continued with an overview of the software structure - describing the programs which reside in low memory and those which reside in extended memory. The chapter concluded by presenting a synopsis of program modules which reside in the LSI FEP (Low Memory) and LSI FEP (High Memory) programs.

CHAPTER 5

SOFTWARE TEST AND EVALUATION

This chapter describes the testing phase of the Software Development Life Cycle (ref para 1.5.1.5). It begins with a short discussion of Testing Methodology and ends with a presentation of the Testing Results.

5.1 Testing Methodology -

Two testing strategies were used during this investigation. A coarse test was conducted using Requirements-based testing [31:185]. A more detailed test was conducted using a Program-based testing [31:195] approach.

5.1.1 Requirements-based Testing -

The traditional requirements-based testing method is functional testing. In functional testing, a program or

software system is viewed as a "black box" which accepts known inputs, applies the relevant function to these inputs, and generates outputs [31:185]. It is usually applied over a range of classes of input data and typically delivers output belonging to one of a number of different possible classes.

A rigorous functional test would include selecting the input data classes, identifying extreme cases and boundary conditions, and establishing the class categories for the expected output. The successful accomplishment of these steps pre-supposes the existence of concise and detailed requirements and specification documents.

Due to the general nature of the FEP Software Requirements Analysis (ref Chapter 2 and Appendix A), such a rigorous functional test could not be conducted. The system functional requirements were presented in a descriptive rather than quantitative manner. Furthermore, the full set of software requirements (Appendix A) was consolidated into a summary table (Table 1-1) for easier assimilation. This summary table was then further condensed into the page length System Level Requirements Table (Table 2-1) upon which the requirements prioritization (ref para 2.2) and software design (ref Chapter 4) were based.

Therefore, it is this final capsule-format of the

functional requirements which the functional test would be conducted against. Since this generalized format precluded an in-depth functional test, the goals of the functional test were altered.

Rather than test implemented functions at a microscopic level, it was decided that the test should be conducted at a macroscopic level. The resulting purpose of the test was to identify the presence, absence, and well-being of the functions displayed in Table 2-1. In other words, rather than providing concise quantitative results, the test would produce a high level survey of implementation completeness. This test was performed mostly by inspection and it identified functions (ref Table 5-1) which were:

a. implemented with no obvious high-level limitations
b. implemented with observable high-level limitations
c. not implemented (due to prioritization decisions)

5.1.2 Program-based Testing -

One of the weaknesses of requirements testing is its failure to test computational features of a program which are related to the design and implementation of the program and which are not a part of its requirements [31:195]. Program-based testing involves the selection of test data which tests specific computational structures of the

program. The most widely studied program-based testing methods are those that involve the selection of test data which causes the execution of specific statements, branches, or paths of the program. These methods are referred to as "structured testing methods" [31:195].

5.1.2.1 Branch Testing - Branch testing was the earliest form of structured testing to be studied and systematically applied to the testing of programs [31:196]. The technique requires that test data be constructed that causes each branch in a program to be transversed at least once.

5.1.2.2 Statement Testing - A more restricted kind of structured testing, statement testing, requires that each statement in a program be executed at least once on some test [31:196].

5.1.2.3 Path Testing - Several studies of the effectiveness of branch testing indicates that there are large numbers of errors whose existence is not necessarily revealed by the testing of all branches in a program [31:199]. Many of these errors are related to combinations of branches and are revealed only by a test that causes a program path to be followed which contains the combination. Path testing requires that every "logical path" through a program should be tested at least once. The difficulty with this idea is that a program which contains loops will, in

general, have an infinite number of possible paths [31:199].

5.2 Testing Results -

Both requirements-based (black box) and program-based (white box) testing were performed. As discussed previously, the black box testing was conducted as a survey approach to assessing the functional completeness of the LSI FEP software implementation.

Branch testing was chosen as the program-based testing strategy because it enabled a thoroughness beyond that of statement testing, yet avoided the prohibitively expensive effort of a comprehensive path testing. It was complemented throughout the software development process by informal code walk-throughs conducted by the author.

5.2.1 Black-box Testing Results -

Table 5-1 indicates that all of the Top priority, High priority, and Medium priority (ref para 2.2) software requirements have been implemented to some extent. The following paragraphs describe those requirements in Table 5-1 which failed the requirements-based testing.

5.2.1.1 Software on Each Processor - Although the scope of this thesis project specifically limited the implementation

Table 5-1 Test Plan Procedures and Results

NMBR	REQUIREMENT	PRI	TEST CRITERIA	P	F	REMARKS
1.1	Two Processors		VAX and LSI hardware in place.	P		
1.2	Communications Link		Able Computer Tech. Interlinks in place.	P		
1.3	Software on each processor	T	FEP software in place on both machines		F	VAX FEP software not implemented
2.1	Multi-programmed environment (Host O/S)		VAX/VMS O/S	P		
2.2	Mass storage (Host O/S)		VAX/VMS O/S	P		
2.3	Comm link support (Host O/S)		VAX/VMS O/S		F	no DB-11B device driver
2.4	High level language support (Host O/S)		VAX 'C' compiler	P		
3.1	Multi-terminal support (LSI O/S)		RT11XM O/S	P		
3.2	Mass storage (LSI O/S)		RT11XM O/S	P		
3.3	Comm link support (LSI O/S)		RT11XM O/S	P		

Table 5-1 Test Plan Procedures and Results (Cont'd)

NMBR	REQUIREMENT	PRI	TEST CRITERIA	P	F	REMARKS
3.4	High level language support (LSI O/S)		Telecon 'C' compiler	P		
4.1	Provide 'single-user environment	H	good response, no data contamination	P		
4.2	Consistent with VAX/VMS operation	H	no new Host interface requirements	P		
4.3	Procedural assistance	L	'Help' facility in place		F	Not implemented
4.4	Easy to learn and use	M	no new user interface requirements	P		
4.5	Pocessing support invisible to users	M	no new user interface requirements	P		
5.1.1	Power source compatibility		functional within DEL environment	P		
5.1.2	Temperature range compatibility		functional within DEL environment	P		
5.1.3	Humidity range compatibility		functional within DEL DEL environment	P		
5.2.1	Unattended operation	H	Operator intervention not required	P		

Table 5-1 Test Plan Procedures and Results (Cont'd)

NMBR	REQUIREMENT	PRI	TEST CRITERIA	P	F	REMARKS
5.2.2	Support for 7 terminals	H	software and hardware in place	P		
5.2.3	Support for a serial line printer	L	software and hardware in place		F	Not implemented
5.2.4	Support for LCN study	M	'LSIFEP.DAT' linkage and comm in place	P		Last block not written to disk
5.2.5	DELNET integration	L	DELNET linkage and comm in place		F	Not implemented
6.1	In-house maintenance support		DEL technician hardware familiarity	P		
6.2.1	Modular software expansion support	M	Top-down, structured implementation	P		
6.2.2	Physical configuration expansion support	M	display/modification of system status		F	only displays system status
7.1	On-hand component use to minimize cost		no new expenditures during this effort	P		

Table 5-1 Test Plan Procedures and Results (Cont'd)

Test Plan Procedures and Results Key

NMBR : Functional Requirement number (ref Table 2-1)
REQUIREMENT : Discription of the requirement matching NMBR
PRI : Implementation Priority (ref para 2.2)
[NOTE: only software functions have been assigned priorities]
T = Top (ref para 2.2.1)
H = High (ref para 2.2.2)
M = Medium (ref para 2.2.3)
L = Low (ref para 2.2.4)
TEST CRITERIA : Pass / Fail criteria
P : Test results : Pass
F : Test results : Fail
REMARKS : Limitations or reasons for test failure

effort to LSI FEP software only (ref para 1.4), this requirement was deemed an initial failure because the system could not function properly without VAX FEP software being implemented on the host.

5.2.1.2 Comm Link Support (Host O/S) - Neither DEC nor Able Computer Technology, Inc provided a VAX/VMS software device driver to support the DR-11B Direct Memory Access (DMA) board. This deficiency provided the single critical limiting factor in the FEP implementation and testing.

[Note: A possible DR-11B driver has been located at GD Serle (Illinois) and efforts are underway to procure a copy of this driver.]

Without a working driver on the host end of the comm link, most of the LSI FEP comm link software could not be tested (ref para 2.2.1). The purpose of a network protocol is to provide mutually agreed upon handshaking between cooperating computers. This goal cannot be realized when the distant end (host) is not capable of handshaking.

Although Transport Layer protocol (ref para 3.3.4) was implemented, no attempt was made to code the Data Link Layer protocol (ref para 3.3.2). It was not clear that the latter protocol was really required and the lack of a host driver provided no resolution of the issue.

5.2.1.3 Procedural Assistance - This function was not implemented due to the time constraints placed upon the project and the relatively low priority placed upon this requirement (ref para 2.2.4.1).

5.2.1.4 Support for a Line Printer - This function was not implemented for the same reasons given in para 5.2.1.3 (ref para 2.2.4.2).

5.2.1.5 DELNET Integration - This function was not implemented for the same reasons given in para 5.2.1.3 (ref para 2.2.4.3).

5.2.1.6 Physical Configuration Expansion Support - This function was envisioned to have included real-time control over configuration modification [1:67]. The actual implementation, however, does not allow the operator to interactively modify any system parameter or database status variable.

This function is minimally addressed with the operator ability to display the status of select database tables and variables (ref para 4.4.1.17 and Appendix F).

5.2.2 White-box Testing Results -

Branch testing was applied to all branches of the program which could be reached using keyboard inputs. These branches included all of the terminal concentrator functions

(ie. - interrupt handling, buffering, output, etc.). However, the DMA servicing code could not be reached nor tested due to the host's inability to generate or receive DMA traffic.

5.2.2.1 Support for LCN Study - Although this function passed its testing (ref Table 5-1), one limitation was revealed. This limitation appears to be caused by a coding deficiency in the LFMLHI.C library program. This program (ref para 4.3.2.2) contains the 'C' run-time utilities used for file manipulation --- among which, is the "fclose" subroutine used to close the "LSIFEP.DAT" accounting file.

This subroutine fills a memory buffer with data intended for a disk file. When the buffer is full, it is flushed and written to disk. During normal processing, this presents no problem. However, upon system termination, the "LFEPLO.C" program issues the "fclose" subroutine call to close the accounting file (ref para 4.4.1.24). Testing revealed that the "fclose" subroutine does not flush the partially filled memory buffer prior to closing the file. Therefore, any recorded data which is memory resident at system termination will not be written to disk.

Another problem may exist during accounting file disk writes. The source code ensures that a large enough block of free disk area is available before opening the

"LSIFEP.DAT" file. However, this file grows dynamically and its size depends upon terminal traffic intensity. It is not readily apparent what happens when the bottom of the "LSIFEP.DAT" file butts up against the front of an existing file. Further investigation should be conducted to validate that the "LSIFEP.DAT" file does not grow without bound --- and there-by overwrite any existing files.

5.3 Summary -

This chapter described the testing methodologies used to validate the LSI FEP software. In it, the requirements-based testing and program-based testing strategies were defined. The results of both testing approaches on the LSI FEP software were discussed. Limitations of the testing phase were described and generally attributed to the lack of a VAX/VMS supported device driver for the DMA interface.

One potential source for the driver had been located and efforts were underway to procure the driver from that source.

CHAPTER 6

CONCLUSION

6.1 The Problem Revisited -

The problem investigated during this project was to design, implement, and document the LSI-11/23 portion of the Communications Front End Processor (FEP) system (ref para 1.3).

This investigation began with an analysis of the functional requirements which included prioritizing the requirements for the purpose of ordering the implemention effort. Design decisions and tradeoffs were examined in the light of these priority assignments. Next, network design issues were discussed using the ISO Reference Model as a departure point for the protocol layers. Software design and implementation issues were then considered with

capabilities and limitations examined and the software structure defined.

This investigation ended with a test and evaluation phase conducted using both requirements-based and program-based testing methods.

6.2 Accomplishments -

Specific accomplishments of this thesis effort can be classified as either hardware or software improvements.

6.2.1 Hardware Improvements -

The major hardware improvement was the physical placement of the DMA interlink boards within the VAX-11/780 and LSI-11/23 computers and connecting these boards via the DMA cabling. Other accomplishments included insertion of the four serial port DLV11-J card (ref para 2.3.4) and replacement of the two standard M8044 Plessey memory boards (together addressing memory locations 0 - 377777 octal) with a single MSV11-E [20:468-487] card which allowed full LSI-11/23 addressing from 0-777777 octal.

6.2.2 Software Improvements -

Software improvements included accomplishment of the middle phases - Design, Coding, and Testing - of the Software Development Life Cycle model (ref para 1.5.1). These phases proceeded from the Requirements Analysis and Specification phases accomplished in a previous [1] thesis effort.

Specific software enhancements included the coding and testing of the Terminal Concentration (ref Chapter 4) features as well as the Accounting data recording features for the Local Computer Network study requirement. Code was written to service the LSI-11/23 end of the DMA interface, but resources were not available to test this software.

6.3 Discussion -

This section summarizes the thesis investigation from the point of view of designer and implementer. Presentation chronacles the problems (and resolutions) that occurred at various points.

6.3.1 Scope -

The first hard decision to be made was limiting the scope of this investigation to implementing the LSI-11/23

portion of the FEP system. There is a tendency for every builder to want to create an entity which is complete unto itself. It is very difficult to confine oneself to building a piece of some whole that will not be realized for some time. This project provided the challenge of paring the FEP implemetation down to a size for which a realistic effort could be expected to produce a reasonable chance of success.

6.3.2 Requirements Prioritization -

One of the early major decisions concerned assigning certain requirements into the low priority category (ref para 2.2.4). Early expectations were that all but the low priority functions would be implemented. Tagging a function as low priority was, in effect, passing-the-buck for its ultimate implementation to a follow-on thesis investigation. It was paramount, therefore, that the "nice to have" functions be properly identified and placed in the low priority class.

6.3.3 Design Decisions and Tradeoffs -

Decisions made immediately after function priority selection shaped and molded the rest of the implementation effort. These design decisions provided the guiding framework which gave eventual direction to the software coding. Each decision funneled the next issue into an increasingly narrow path toward resolution. It was

imperative that the early design decisions be correct because flexibility for change grew smaller with each successive decision.

6.3.4 Network Design and Protocol Issues -

The network design was accomplished at too early a stage in the project. It proceeded from a traditional network approach and resulted in what may be an over-designed network. The primary reason for this was that, at this early point in the project, the capabilities and requirements of the Able Computer Technology, Inc. Interlink DMA interface were not fully understood.

Even now, any increased knowledge felt by the author in this respect continues to exist only as a subjective opinion which cannot be put to the test until a host device driver becomes operationally available. For this reason, Chapter 3 was allowed to remain as written and its evaluation deferred pending arrival of the testing tools.

6.3.5 Software Design and Implementation -

Chapter 4 chronicled a period of time in which author experienced the alternating extremes of exhilarating satisfaction and nagging frustration. In several instances, critical code fragments that were expected to require many rewrites worked flawlessly the first time. On other

occasions, errors which were as dumb as they were transparent delayed the project for days.

6.3.6 Software Test and Evaluation -

It seems that no matter how strenuously one advocates testing as a cycle-long requirement, it always seems to be deferred until finally addressed at the eleventh hour of a software project. Although top- down implementation of the code segments protects against this to a large degree, it does not completely eliminate this panic mode of testing. Throughout this implementation, testing seemed to lag behind production, followed only by documentation in the race of procrastinations.

6.4 Recommendations -

Several concrete recommendations emerged from this study as follows:

6.4.1 DR-11B Device Driver Installation -

Without a working device driver on the host (VAX) end of the DMA link, this project would seem to be incomplete. As stated in Chapter 5, a potential DR-11B device driver has been located and and a copy of the driver source code has been requested from the writer. This driver currently only

handles word mode data transfers for the VAX-11/780 and would have to be modified to provide the expanded capabilities of block mode transfers for the FEP application.

This modification does not appear to be a trivial exercise. A firm grasp of the VAX assembly instruction set and I/O data base will be required as well as a good understanding of the DMA programming.

The driver should be examined upon arrival and modified by a qualified programmer to address the needs of the VAX FEP.

6.4.2 Data Link Protocol -

As stated in Chapter 5, the Data Link Protocol was not implemented primarily due to the uncertainty of its requirement. This aspect should be further investigated once the DR-11B driver becomes operational.

6.4.3 Buffer Sizing -

The program buffer sizes (200 characters for terminal traffic and 2000 characters for node traffic) were chosen arbitrarily (ref para 3.3.3.3 and para 3.3.4.1). These sizes may be adjusted based upon further testing.

6.4.4 Number of Terminals -

At present, the LSI FEP will support seven dedicated VT-100 interactive terminals and one VT-100 terminal which can operate as the SOC or as the eighth interactive terminal. The requirements specify that this configuration should be expandable to sixteen terminals. The LSI-11/23 unibus structure should be further investigated to ensure that enough I/O port addresses and interrupt vectors exist for this expansion.

6.4.5 LFMLHI.C File Manipulation Limitation -

As discussed in Chapter 5, the "fclose" function does not flush the memory buffer of data destined for disk writing prior to closing the file. This deficiency should be corrected. At the same time, the potential 'write without bound' question posed in Chapter 5 should also be investigated.

6.4.6 The Completed DEL FEP -

Finally, the whole to which this thesis effort is only a part should be concluded. Namely, the remaining VAX FEP software should be designed and implemented. In addition, the low priority tasks (deferred from implementation in this thesis) should be re-examined and implemented.

BIBLIOGRAPHY

1. Gnadt, Larry Wm. Design and Implementation of a Front-End Processor System for a Digital Equipment Corporation VAX-11/780. MS Thesis; Air Force Institute of Technology, School of Engineering, Wright-Patterson AFB, Ohio, 1982.

2. Plessey Peripheral Systems. General Information. Users Manual MA-703525 Rev A; Irvine, California; 8 May 1980.

3. Plessey Peripheral Systems. PM-MFV11A Multifunction Board Manual. Users Manual MA-703510 Rev A; Irvine, California; April 1981.

4. Digital Equipment Corporation. Microcomputer Interfaces Handbook. Reference Manual; 1980.

5. Tanenbaum, Andrew S. Computer Networks. Englewood Cliffs, New Jersey; Prentice-Hall, Inc. 1981.

6. Seward, Walter D. Lecture material distributed in EE6.54 (Computer Communications Networks) and in EE7.54 (Advanced Topics in Computer Networks). School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio. 1983.

7. Digital Equipment Corporation. RT-11 Software Support Manual, Order No. AA-H379A-TC; Computer software documentation. Maynard, Massachusetts. 1981.

8. Digital Equipment Corporation. Introduction to RT-11, Order No. AA-5281B-TC; Computer software documentation. Maynard, Massachusetts. 1981.

9. Digital Equipment Corporation. RT-11 System Users Guide, Order No. AA-5279B-TC; Computer software documentation. Maynard, Massachusetts. 1981.

10. Digital Equipment Corporation. RT-11 System Messages Manual, Order No. AA-5284C-TC; Computer

software documentation. Maynard, Massachusetts. 1981.

11. Digital Equipment Corporation. RT-11 Master Index, Order No. AA-H380A-TC; Computer software documentation. Maynard, Massachusetts. 1981.

12. Digital Equipment Corporation. RT-11 Documentation Directory, Order No. AA-5285F-TC; Computer software documentation. Maynard, Massachusetts. 1981.

13. Digital Equipment Corporation. PDP-11 Keypad Editor User's Guide, Order No. AA-H583A-TC; Computer software documentation. Maynard, Massachusetts. 1981.

14. Digital Equipment Corporation. RT-11 Programmer's Reference Manual, Order No. AA-H378A-TC; Computer software documentation. Maynard, Massachusetts. 1981.

15. Digital Equipment Corporation. PDP-11 MACRO-11 Language Reference Manual, Order No. AA-5075B-TC; Computer software documentation. Maynard, Massachusetts. 1981.

16. Digital Equipment Corporation. RT-11 System Release Notes, Order No. AA-5286D-TC; Computer software documentation. Maynard, Massachusetts. 1981.

17. Digital Equipment Corporation. RT-11 Installation and System Generation Guide, Order No. AA-H376A-TC; Computer software documentation. Maynard, Massachusetts. 1981.

18. Able Computer Technology, Inc. Preliminary User's Guide to INTERLINK/LSI; Computer hardware documentation. Irvine, California. March 1980.

19. Able Computer Technology, Inc. Preliminary User's Guide to INTERLINK/UNI; Computer hardware documentation. Irvine, California. April 1979.

20. Digital Equipment Corporation. Microcomputers and Memories; Handbook. Maynard, Massachusetts. 1981.

21. TELECON Systems. TELECON 'C' Users Manual. Software documentation. San Jose, California.

22. Kernighan, Brian W. and Dennis M. Ritchie. The 'C' Programming Language. Prentiss Hall Software Series Textbook. Prentice-Hall, INC. Englewood Cliffs, New Jersey. 1978.

23. Hartrum, Thomas C. 'C' Compiler User's Manual. Course Handout. Air Force Institute of Technology. Wright-Patterson Air Force Base. Dept of Electrical Engineering. 1982

24. Digital Equipment Corporation. Programming in VAX-11 'C'. Software manual. Order No. AA-L370A-TE. Maynard, Massachusetts. May, 1982.

25. Zelkowitz, Marvin V. "Perspectives on Software Engineering", ACM Computing Surveys. Vol. 10, No. 2 (pp. 197-216). June 1978.

26. Boehm, B., McClean, R., and Urfrig, D. "Some Experience with Automated Aids to the Design of Large Scale Reliable Software." International Conference on Reliable Software. (pp. 105-113). ACM, New York. 1975.

27. Hamilton, M. and Zeldin, S. "Higher Order Software - a Methodology for Defining Software". IEEE Transactions on Software Engineering. Vol. 2, No. 1 (pp. 9-32). March 1976.

28. Peters, Lawrence J. Software Design: Methods and Techniques. Yourdon Press. New York. 1981.

29. Krieger, M.S. and P. J. Plauger. "C Language's Grip on Hardware Makes Sense for Small Computers." Electronics. May 8, 1980.

30. Whitesmiths, Ltd. Whitesmiths Software Catalogue. Spring 1981 edition. Whitesmiths, Ltd. New York, New York. 1981.

31. Howden, William E. "A Survey of Dynamic Analysis Methods." IEEE Tutorial on Software Testing and Verification Techniques. pp 185-206. 1978.

INDEX

APPENDIX A

SOFTWARE REQUIREMENTS ANALYSIS

This appendix reproduces the Designer Perspective Software Sub-System Requirements indentified in ref [1]. This model served as the departure point upon which the current thesis effort progressed.

Requirement	Description
1	Local Computer Network
1.1	Two Processors
1.2	Communications Link
1.3	Software On Each Processor
1.3.1	Front-End Software
1.3.1.1	Support User Terminals
1.3.1.1.1	Virtual Link
1.3.1.1.1.1	Packet Interlock
1.3.1.1.1.2	Sequence Number
1.3.1.1.2	Information Routing
1.3.1.1.2.1	Logical Link
1.3.1.1.2.1.1	Same Processor
1.3.1.1.2.1.2	Between Processors
1.3.1.1.2.2	Physical Link
1.3.1.1.2.2.1	Between Processors
1.3.1.1.3	Message Assembly/Disassembly
1.3.1.1.3.1	Header
1.3.1.1.3.1.1	Source Terminal
1.3.1.1.3.1.2	Destination Terminal
1.3.1.1.3.1.3	Message Number
1.3.1.1.3.1.4	Message Length
1.3.1.1.3.2	Message Text
1.3.1.1.3.2.1	Alpha-numeric Characters
1.3.1.1.3.3	Transmission Protocol
1.3.1.1.3.3.1	Logical Link
1.3.1.1.3.3.1.1	Source Node
1.3.1.1.3.3.1.2	Destination Node
1.3.1.1.3.3.1.3	Message Number
1.3.1.1.3.3.1.4	Queue Control Information
1.3.1.1.3.3.2	Physical Link
1.3.1.1.3.3.2.1	Source Processor
1.3.1.1.3.3.2.2	Destination Processor
1.3.1.1.3.3.2.3	Message Sequence
1.3.1.1.3.3.2.4	Link Mode
1.3.1.1.3.3.2.5	Queue Length
1.3.1.1.3.3.2.6	Message Space
1.3.1.1.3.3.2.7	Queue Control Information
1.3.1.1.3.3.2.8	Checkfield
1.3.1.1.3.3.2.9	Message Terminator

Requirement	Description
1.3.1.1.4	Link Assignment Strategy
1.3.1.1.4.1	Multiple Links
1.3.1.1.4.2	Multiple Link Types
1.3.1.2	Perform User Tasks
1.3.1.2.1	Operating System Tasks
1.3.1.2.2	Special Functions
1.3.1.3	Comm Link Management
1.3.1.3.1	Control Comm Link
1.3.1.3.1.1	Physical Control
1.3.1.3.1.2	Link Mode
1.3.1.3.1.2.1	Message Mode
1.3.1.3.1.2.2	File Mode
1.3.1.3.2	Assemble Comm Link Message
1.3.1.3.3	Transmit Comm Link Message
1.3.1.3.4	Receive Comm Link Message
1.3.1.3.5	Disassemble Comm Link Message
1.3.1.3.6	Error Check Messages
1.3.2	Host Software
1.3.2.1	Support User Terminals
1.3.2.1.1	Virtual Link
1.3.2.1.1.1	Packet Interlock
1.3.2.1.1.2	Sequence Number
1.3.2.1.2	Information Routing
1.3.2.1.2.1	Logical Link
1.3.2.1.2.1.1	Same Processor
1.3.2.1.2.1.2	Between Processors
1.3.2.1.2.2	Physical Link
1.3.2.1.2.2.1	Between Processors
1.3.2.1.3	Message Assembly/Disassembly
1.3.2.1.3.1	Header
1.3.2.1.3.1.1	Source Terminal
1.3.2.1.3.1.2	Destination Terminal
1.3.2.1.3.1.3	Message Number
1.3.2.1.3.1.4	Message Length
1.3.2.1.3.2	Message Text
1.3.2.1.3.2.1	Alpha-numeric Characters
1.3.2.1.3.3	Transmission Protocol
1.3.2.1.3.3.1	Logical Link
1.3.2.1.3.3.1.1	Source Node
1.3.2.1.3.3.1.2	Destination Node
1.3.2.1.3.3.1.3	Message Number
1.3.2.1.3.3.1.4	Queue Control Information

Requirement	Description
1.3.2.1.3.3.2	Physical Link
1.3.2.1.3.3.2.1	Source Processor
1.3.2.1.3.3.2.2	Destination Processor
1.3.2.1.3.3.2.3	Message Sequence
1.3.2.1.3.3.2.4	Link Mode
1.3.2.1.3.3.2.5	Queue Length
1.3.2.1.3.3.2.6	Message Space
1.3.2.1.3.3.2.7	Queue Control Information
1.3.2.1.3.3.2.8	Checkfield
1.3.2.1.3.3.2.9	Message Terminator
1.3.2.1.4	Link Assignment Strategy
1.3.2.1.4.1	Multiple Links
1.3.2.1.4.2	Multiple Link Types
1.3.2.2	Perform User Tasks
1.3.2.2.1	Operating System Tasks
1.3.2.2.2	Special Functions
1.3.2.3	Comm Link Management
1.3.2.3.1	Control Comm Link
1.3.2.3.1.1	Physical Control
1.3.2.3.1.2	Link Mode
1.3.2.3.1.2.1	Message Mode
1.3.2.3.1.2.2	File Mode
1.3.2.3.2	Assemble Comm Link Message
1.3.2.3.3	Transmit Comm Link Message
1.3.2.3.4	Receive Comm Link Message
1.3.2.3.5	Disassemble Comm Link Message
1.3.2.3.6	Error Check Messages
2	Host Operating System
2.1	Multi-Programmed Environment
2.2	Mass Storage
2.3	Comm Link Support
2.4	High Level Language
3	FEP Operating System
3.1	Support for Maximum Terminal Population
3.2	Mass Storage
3.3	Comm Link Support
3.4	High Level Language

Requirement	Description
4	Consistent User Interface
4.1	Provide "Single User" Environment
4.2	Consistent With VAX/VMS Operation
4.2.1	Single-User/Host Operations
4.2.2	Control/Management Operations
4.2.2.1	Terminal CONNECT
4.2.2.2	Terminal DISCONNECT
4.2.2.3	Command Interpreter
4.3	Procedural Assistance
4.3.1	Single-User/Host Operations
4.3.2	Control/Management Operations
4.3.2.1	HELP Operation
4.4	Easy to Learn and Use
4.4.1	Control/Management Operations
4.4.1.1	HELP Operation
4.4.1.2	Terminal CONNECT
4.4.1.3	Terminal DISCONNECT
4.5	Processing Support Invisible to User
4.5.1	Single-User/Host Operations
4.5.2	Control/Management Operations
5	Operating Environment Compatibility
5.1	Physical Plant Compatibility
5.1.1	Power Source
5.1.2	Temperature Range
5.1.3	Humidity Range
5.2	Academic Compatibility
5.2.1	Unattended Operation
5.2.1.1	Startup Procedure
5.2.1.2	Shutdown Procedure
5.2.1.3	Asynchronous Intermediate Processing
5.2.1.3.1	User Level Messages
5.2.1.3.1.1	Requests
5.2.1.3.1.2	Responses
5.2.1.3.2	System Level Messages
5.2.1.3.2.1	Entries From FEP Console
5.2.1.3.2.2	Responses to System Requests
5.2.1.3.2.3	System Level Status
5.2.1.3.3	Queueing System
5.2.1.3.3.1	Servers: Logical Nodes
5.2.1.3.3.2	Message Movement Strategy
5.2.2	Support for 8 Interactive Terminals
5.2.3	Support for Line Printer

Requirement	Description
5.2.4	Support for Study of LCN
5.2.4.1	Collect Performance Data
5.2.4.1.1	System Level Status
5.2.4.1.1.1	Queue Overflow
5.2.4.1.1.2	Comm Link Errors
5.2.4.1.2	Terminal Session Statistics
5.2.4.1.2.1	Session Number
5.2.4.1.2.2	Terminal Number
5.2.4.1.2.3	Connect Date
5.2.4.1.2.4	Connect Time
5.2.4.1.2.5	Disconnect Date
5.2.4.1.2.6	Disconnect Time
5.2.4.1.2.7	First User File Record
5.2.4.1.2.8	Last User File Record
5.2.4.1.3	User Session Statistics
5.2.4.1.3.1	User Record Number
5.2.4.1.3.2	Session Number
5.2.4.1.3.3	Username
5.2.4.1.3.4	Current System
5.2.4.1.3.5	Logon Time
5.2.4.1.3.6	Logoff Time
5.2.4.1.3.7	Number Messages Input
5.2.4.1.3.8	Number Characters Input
5.2.4.1.3.9	Number Messages Output
5.2.4.1.3.10	Number Characters Output
5.2.4.1.3.11	Number Messages Sent to Printer
5.2.4.1.3.12	Number Characters Sent to Printer
5.2.4.1.3.13	Total Printer Time
5.2.4.1.3.14	Total Number Printers Assigned
5.2.4.1.4	System Queue Statistics
5.2.4.1.4.1	Queue Name
5.2.4.1.4.2	Current Length
5.2.4.1.4.3	Message Number
5.2.4.1.4.4	Event Time
5.2.4.1.4.5	Event Code
5.2.4.2	File Transfer
5.2.4.2.1	Transfer To/From Host
5.2.4.2.2	Disk Media
5.2.4.3	Peripheral Sharing
5.2.4.3.1	Route Output To Printer
5.2.5	DELNET Integration
5.2.5.1	Single-User/DELNET Operations
5.2.5.2	Control/Management Operations

Requirement	Description
6	Supportability
6.1	In-House Maintenance
6.1.1	Hardware
6.1.2	Software
6.2	Expansion
6.2.1	Modular Software
6.2.1.1	Functions
6.2.1.2	Functionally Cohesive
6.2.1.3	Hierarchical Structure
6.2.1.4	Loosely Coupled
6.2.2	Physical Configuration
6.2.2.1	Terminals
6.2.2.2	Processors
6.2.2.3	Comm Links
6.2.3	Inspect Configuration
6.2.4	Modify Configuration
7	Minimum Cost
7.1	On-hand Components
8	Data Security
8.1	No Requirement

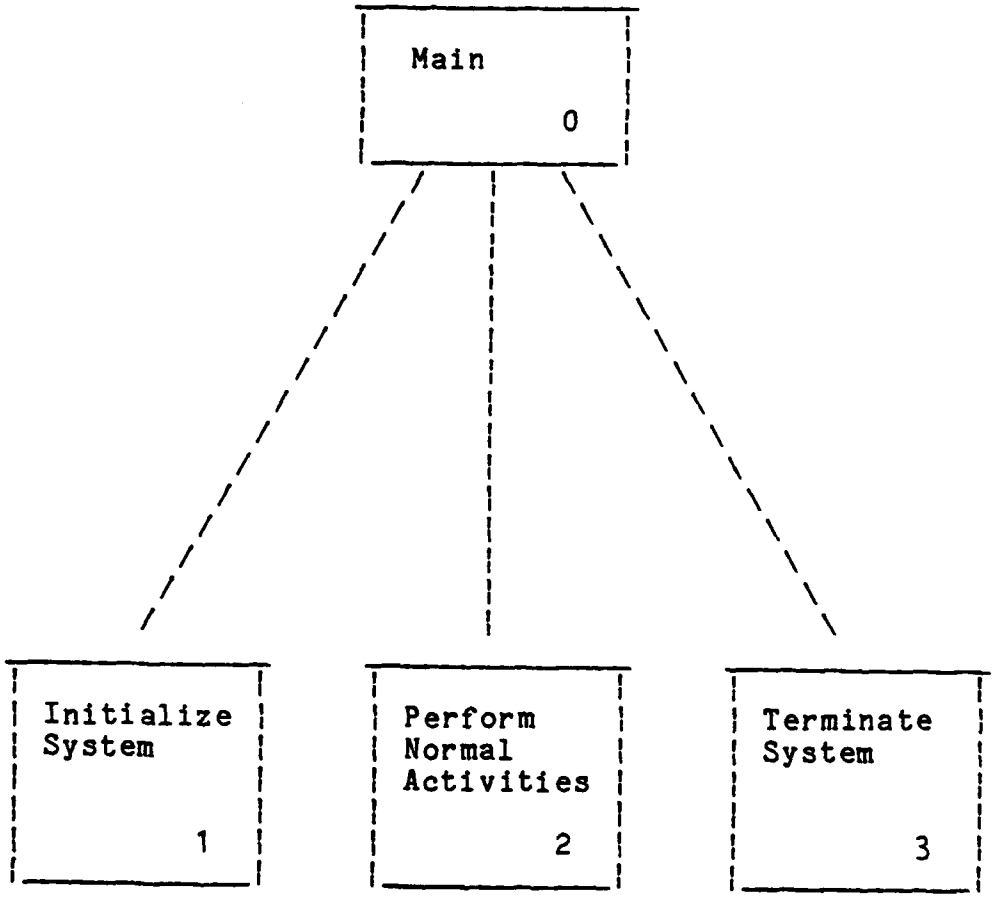
APPENDIX B

LSI FEP STRUCTURE CHARTS

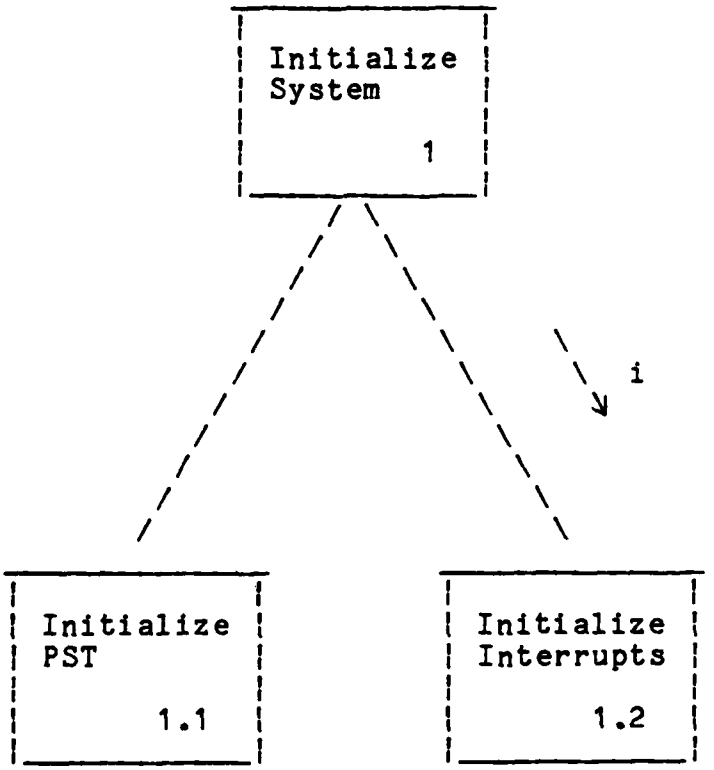
This appendix contains the structure charts used in designing the LSI FEP software modules.

DIRECTORY OF STRUCTURE CHARTS

B.1 Module 0 - Main -

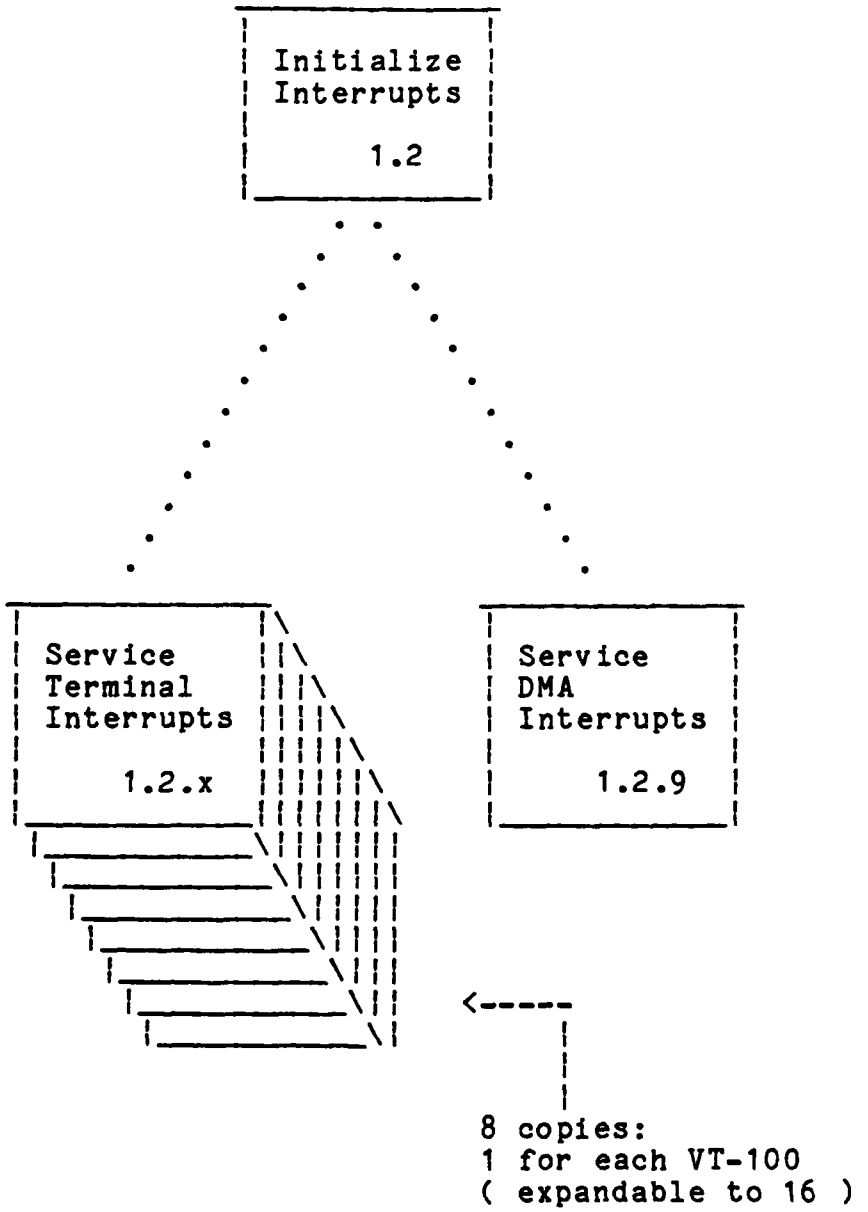


B.2 Module 1 - Initialize System -



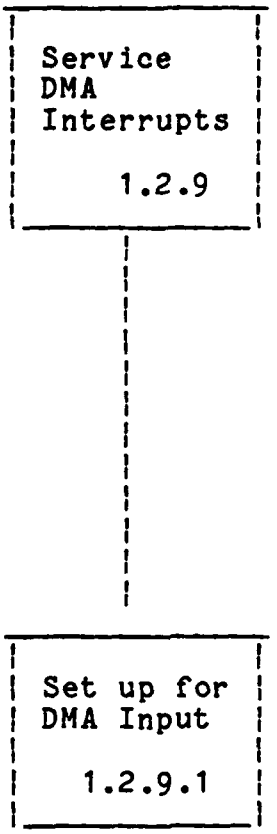
i = index into PST

B.3 Module 1.2 - Initialize Interrupts -

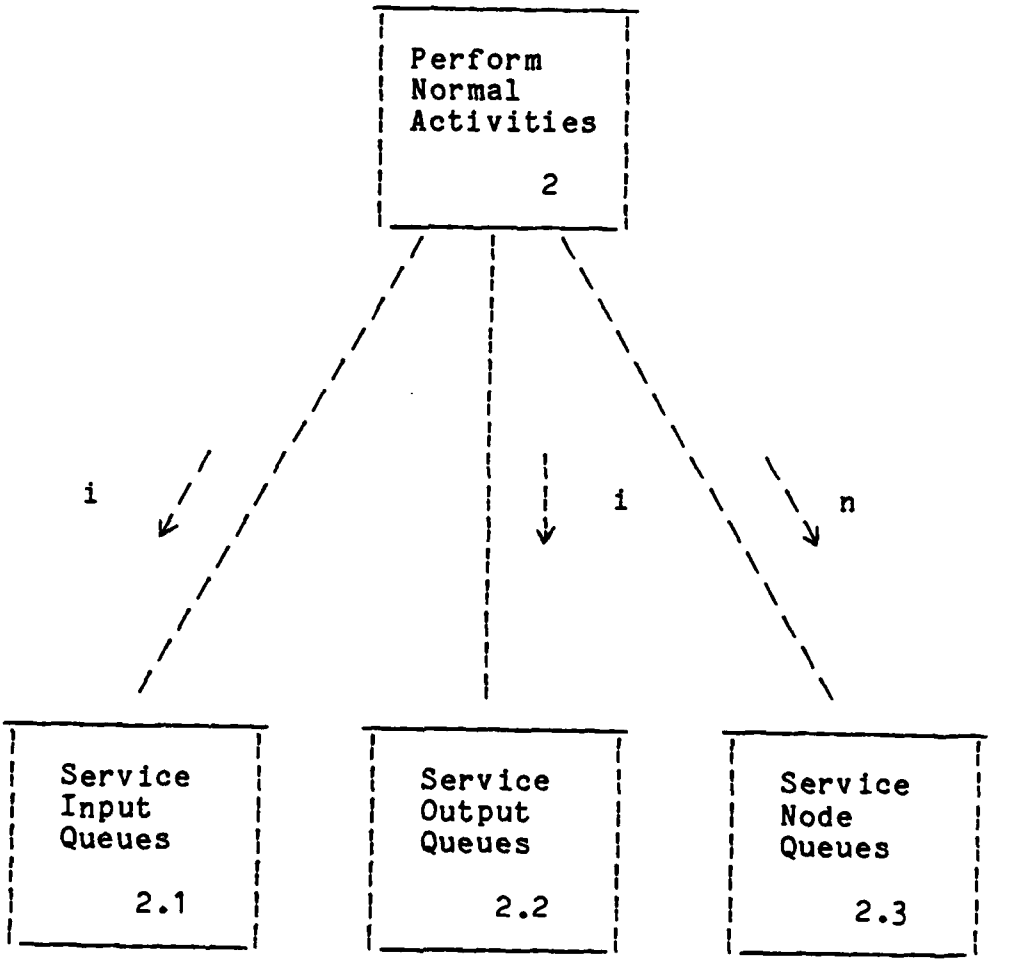


NOTE: Dotted lines indicate that interrupt service routines are ACTIVATED by module 1.2 rather than CALLED directly. Interrupt service routines are INVOKED by the RT11XM operating system upon hardware detection of an I/O interrupt.

B.4 Module 1.2.9 - Service DMA Interrupts -

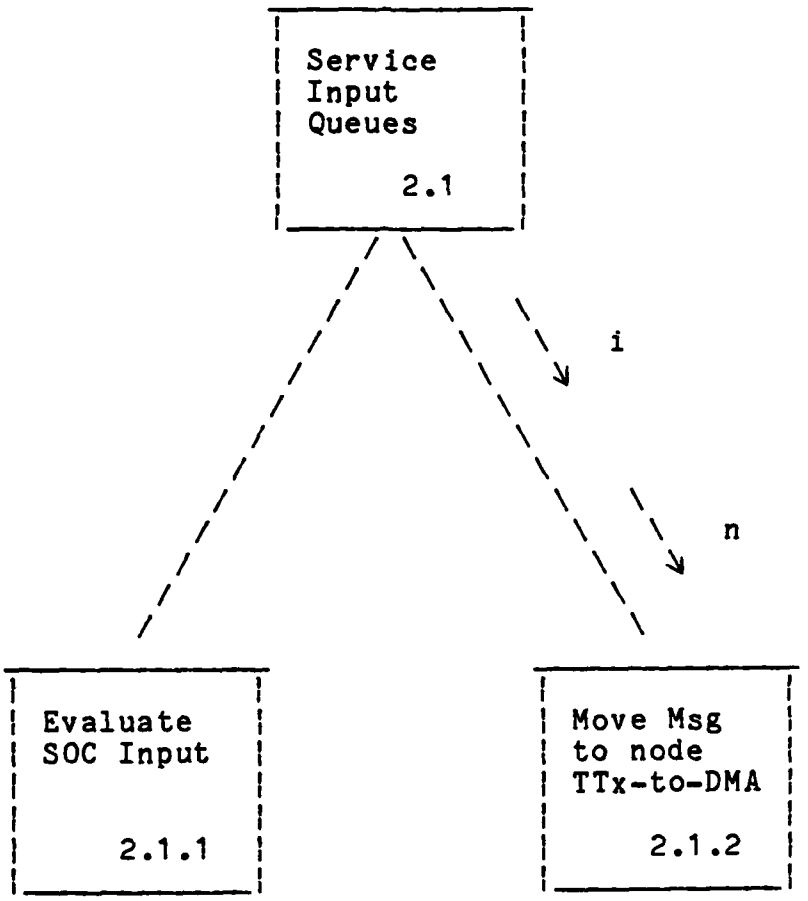


B.5 Module 2 - Perform Normal Activities -



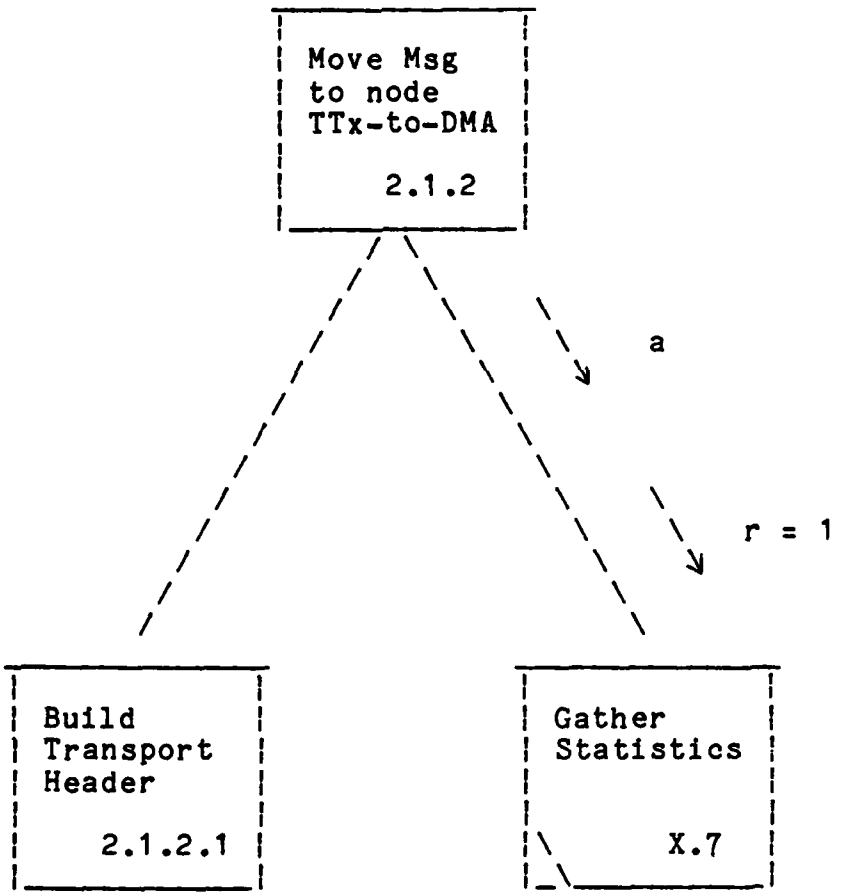
i = index into PST
n = index into NBT

B.6 Module 2.1 - Service Input Queues -



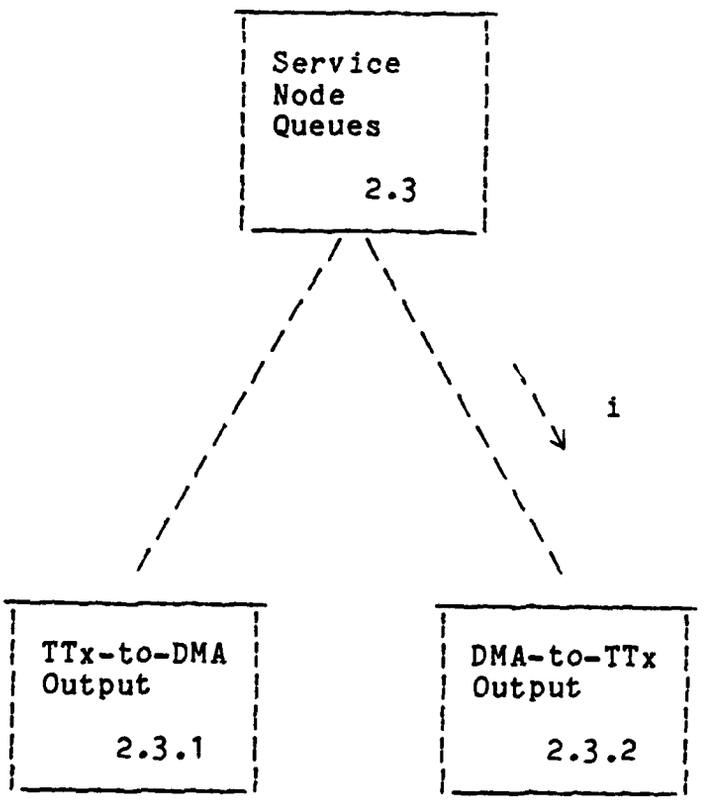
i = index into PST
n = index into NBT

B.7 Module 2.1.2 - Move Message to Node TTxDMA -



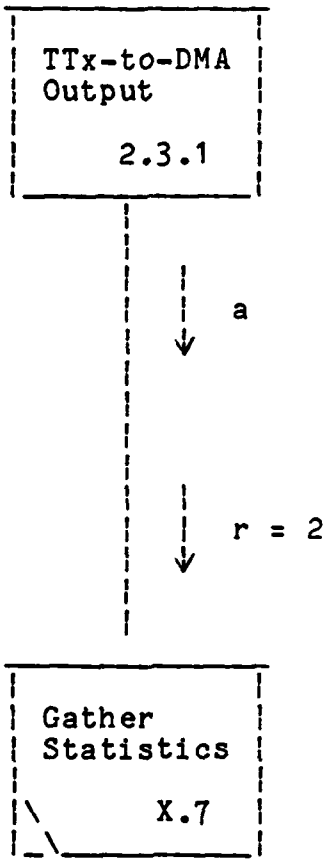
a = address of THT
r = reason for gathering statistics
(1 = entry into a node queue)

B.8 Module 2.3 - Service Node Queues -



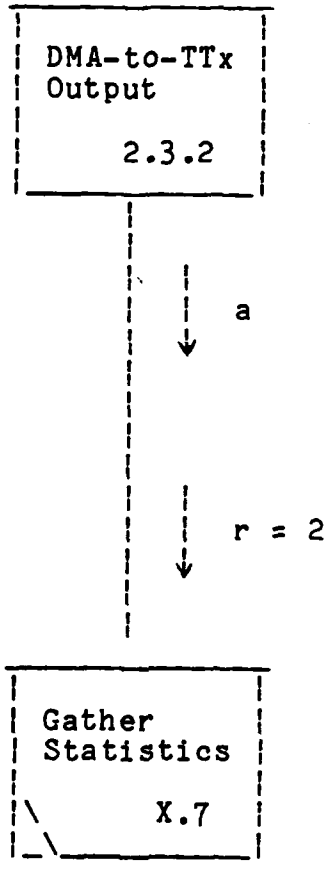
i = index into PST

B.9 Module 2.3.1 - TTx-to-DMA Output -



a = address of message in NodeChar
r = reason for gathering statistics
(2 = exit from a node queue)

B.10 Module 2.3.2 - DMA-to-TTx Output -



a = address of message in NodeChar
r = reason for gathering statistics
(2 = exit from a node queue)

APPENDIX C

LSI FEP DATA DICTIONARY

This data dictionary is divided into two parts. The first part is a dictionary of global data items and structures (Data Item Entries begin on page C-2) from program LFEPLO.C while the second part is a dictionary of functional modules (Functional Modules begin on page C-20) from the same program. Each part contains a short introduction and a table of contents for the section.

C.1 Data Item Entries -

This section contains those data items and entries which are global in nature - that is, they are referenced by more than one program module.

C.1.1 AbortFlag -

Item Name: AbortFlag

Data Type: Boolean Integer

Item Size: 2 bytes

Where written:	Where read:
SOCInterruptServiceRoutine	PerfNormalActivities

Description: AbortFlag is initialized to NO by the 'C' compiler. While it remains equal to NO (where NO = 0), LSIFEP.C continues to execute. When the operator, using the SOC terminal, keys a 'control-C' (^C) input, then the software sets AbortFlag = YES (where YES = 1) and initiates the TermSystem functions.

C.1.2 DMABusyFlag -

Item Name: DMABusyFlag

Data Type: Integer

Item Size: 2 bytes

Where written:	Where read:
InitSystem	TTxToDMAOutput
TTxToDMAOutput	DMAInterruptServiceRoutine
DMAInterruptServiceRoutine	SetUpForInputDMA
SetUpForInputDMA	

Description: DMABusyFlag indicates the status of the DMA channel. Permissable values are as follows:

0 = DMA channel not busy
1 = DMA word (mode) input (VAX to LSI) is expected
2 = DMA block (mode) input (VAX to LSI) is expected
3 = DMA word (mode) output (LSI to VAX) in progress
4 = DMA block (mode) output (LSI to VAX) in progress

C.1.3 DMAwc -

Item Name: DMAwc

Data Type: Integer

Item Size: 2 bytes

Where written:	Where read:
DMAInterruptServiceRoutine	DMAInterruptServiceRoutine SetUpForInputDMA

Description: DMAwc is used to store the word count of the current DMA block transfer.

C.1.4 EndIdx -

Item Name: EndIdx

Data Type: Integer

Item Size: 2 bytes

Where written:	Where read:
SrvNodeQueue	TTxtoDMAOutput DMAtoTTxTransfer

Description: EndIdx is used as an index into the NBT to mark the array element offset of the last character to be moved in the current message.

C.1.5 FileOpenFlag -

Item Name: FileOpenFlag

Data Type: Integer

Item Size: 2 bytes

Where written:	Where read:
InitSystem	

Description: FileOpenFlag is a boolean flag that describes the status of the file open attempt on the accounting statistics file. Possible values include:

NO = 0 = file is not open
YES = 1 = file is open

C.1.6 GoDMAFlag -

Item Name: GoDMAFlag

Data Type: Boolean Integer

Item Size: 2 bytes

Where written:	Where read:
SrvInputQueue EvalSOCInput	SrvInpuQueue

Description: GoDMAFlag is a boolean flag which is used to determine whether a DMA transfer is to occur. A DMA transfer will not occur when a system status request is made from the SOC terminal.

C.1.7 InChar -

Item Name: InChar

Data Type: array

Item Size: 1800 bytes

Where written:	Where read:
SOCInterruptServiceRoutine	SrvInputQueue
T1InterruptServiceRoutine	EvalSOCInput
T2InterruptServiceRoutine	MoveMsgtoNodeTTxDMA
T3InterruptServiceRoutine	
T4InterruptServiceRoutine	
T5InterruptServiceRoutine	
T6InterruptServiceRoutine	
T7InterruptServiceRoutine	
SetUpForInputDMA	

Description: InChar is an 1800 element array of characters. InChar is partitioned by each of the 9 entries in PST yielding an input buffer size of 200 chars for each port.

C.1.8 LastMsgSeqNbr -

Item Name: LastMsgSeqNbr

Data Type: Integer

Item Size: 2 bytes

Where written:	Where read:
BldTransportHeader	BldTransportHeader

Description: LastMsgSeqNbr is a variable which equals the last message sequence number assigned to an outbound DMA transfer. When this number, incremented for each new message, equals a predefined upper limit, it is reset to zero.

C.1.9 NBT -

Item Name: NBT

Data Type: structure

Item Size: 24 bytes

Where written:	Where read:
InitSystem	InitSyaytem
PerfNormalActivities	PerfNormalActivities
MoveMsgtoNodeTTxDMA	SrvInpuQueue
SrvNodeQueue	MoveMsgtoNodeTTxDMA
TTxtoDMAOutput	BldTransportHeader
DMAtoTTxTransfer	SrvNodeQueue
	TTxtoDMAOutput
	DMAtoTTxTransfer

Description: The Node Buffer Table (NBT) is a global communication structure through which several subroutines can manage the NodeChar buffer transactions.

C.1.10 NBTCharCount -

Item Name: NBTCharCount

Data Type: Integer

Item Size: 2 bytes

Where written:	Where read:
BldTransportHeader	SrvInputQueue BldTransportHeader SrvNodeQueue

Description: NBTCharCount is the message character count used for messages residing in character array NodeChar. NBTCharCount is always an even number because the DMA transfer protocol requires word (2 bytes) transfers and messages residing in NodeChar are either headed for the DMA or have just been received via DMA.

C.1.11 NodeChar -

Item Name: NodeChar

Data Type: array

Item Size: 4000 bytes

Where written:	Where read:
MoveMsgtoNodeTTxDMA	SrvNodeQueue TTxtoDMAOutput DMAtoTTxTransfer

Description: NodeChar is a 4000 element character array which contains complete messages headed for (or received from) the DMA interface. It is partitioned by the 2 entries in NBT into 2000 character buffers, one for each of the 2 node queues. Pointers into NodeChar are maintained in the structure NBT.

C.1.12 OutChar -

Item Name: OutChar

Data Type: array

Item Size: 1800 bytes

Where written:	Where read:
DMAtoTTxTransfer	SrvOutputQueue

Description: OutChar is an 1800 element character array partitioned by the 9 entries of the PST into 200 character buffers for each of the 9 ports. OutChar contains characters to be displayed upon the terminal screen of the respective port console. OutChar pointers are maintained in structure PST.

C.1.13 PST -

Item Name: PST

Data Type: structure

Item Size: 604 bytes

Where written:	Where read:
InitSystem	InitSystem
InitPST	InitPST
InitInterrupts	InitInterrupts
PerfNormalActivities	PerfNormalActivities
EvalSOCInput	SrvInpuQueue
MoveMsgtoNodeTTxDMA	EvalSOCInput
SrvOutputQueue	MoveMsgtoNodeTTxDMA
DMAtoTTxTransfer	BldTransportHeader
SOCInterruptServiceRoutine	SrvOutputQueue
T1InterruptServiceRoutine	SrvNodeQueue
T2InterruptServiceRoutine	DMAtoTTxTransfer
T3InterruptServiceRoutine	TermSystem
T4InterruptServiceRoutine	SOCInterruptServiceRoutine
T5InterruptServiceRoutine	T1InterruptServiceRoutine
T6InterruptServiceRoutine	T2InterruptServiceRoutine
T7InterruptServiceRoutine	T3InterruptServiceRoutine
SetUpForInputDMA	T4InterruptServiceRoutine
	T5InterruptServiceRoutine
	T6InterruptServiceRoutine
	T7InterruptServiceRoutine
	SetUpForInputDMA

Description: The Port Status Table (PST) is 9 entry global communication structure through which several subroutines can manage the InChar and OutChar buffer transactions. Also, PST contains the port addresses of each port as well as the addresses of the interrupt vectors and interrupt service routines.

C.1.14 PSTCharCount -

Item Name: PSTCharCount

Data Type: Integer

Item Size: 2 bytes

Where written:	Where read:
SrvInputQueue	SrvInputQueue MoveMsgtoNodeTTxDMA

Description: PSTCharCount is a character count representing the size (in bytes) of an input request. This size includes the number of characters typed on the keyboard as well as the size of the Transport Header which is appended to the front of the message prior to message movement into the TTxDMA node queue. PSTCharCount (unlike NBTCharCount) may be an odd number.

C.1.15 StartIdx -

Item Name: StartIdx

Data Type: Integer

Item Size: 2 bytes

Where written:	Where read:
SrvInputQueue	SrvInputQueue EvalSOCInput

Description: StartIdx is a temporary variable used primarily to calculate, along with StopIdx, the size (character count) of the current input request from a terminal operator. It is set equal to the current position of the PST InChar 'get' index and remains unchanged (while the 'get' index increments) throughout processing of the current request.

C.1.16 StopIdx -

Item Name: StopIdx

Data Type: Integer

Item Size: 2 bytes

Where written:	Where read:
SrvInputQueue	SrvInputQueue EvalSOCInput

Description: StopIdx is a temporary variable used primarily to calculate, along with StartIdx, the size (character count) of the current input request from a terminal operator. It is initially set equal to StartIdx and increments as each character in InChar is scanned. When a carriage return is encountered in InChar, StartIdx is subtracted from StopIdx to yield the number of characters in the input string.

C.1.17 THT -

Item Name: THT

Data Type: structure

Item Size: 36 bytes

Where written:	Where read:
BldTransportHeader	MoveMsgtoNodeTTxDMA SrvNodeQueue

Description: The Transport Header Table (THT) is a 36 character structure used as a template to form the message Transport Header (para.) which is appended to the beginning of each DMA-bound input request.

C.2 Functional Modules -

This section contains a brief PDL description of the processing logic in each LFEPLO.C functional module.

C.2.1 Main -

Called From: 'C' Shell

Modules Called:
1 InitSystem
2 PerfNormalActivities
3 TermSystem

Globals Read: N/A

Globals Written: N/A

PDL Description:

CALL InitSystem to initialize the system

CALL PerfNormalActivities to conduct normal activities

CALL TermSystem to effect an orderly return to RT11XM

C.2.2 InitSystem -

Module Number: 1

Called From: 0 Main

Modules Called: 1.1 InitPST
1.2 InitInterrupts

Globals Read: NBROFBUFFERS
NBROFPORTS
NBT
NODEBUFFERSIZE
PST
TERMBUFFERSIZE

Globals Written: DMABusyFlag
FileOpenFlag
NBT
PST

PDL Description:

```
set DMA Busy Flag = not busy status

if the accounting file can be opened
    set the File Open Flag = File is open
else
    set the File Open Flag = File is not open
    display an error message to the SOC

for all 9 entries in the Port Status Table
    initialize the 4  InChar indexes
    initialize the 4 OutChar indexes

for both entries in the Node Buffer Table
    initialize the 4 NodeChar indexes

designate NBT entry 0 as "TTx"
designate NBT entry 1 as "DMA"
fetch the addrs of the Interrupt Service Routines

CALL InitPST ()  --- to further initialize the PST

for all 9 I/O ports
    CALL InitInterrupts () --- to activate interrupts
```

C.2.3 InitPST -

Module Number: 1.1

Called From: 1 InitSystem

Modules Called: N/A

Globals Read: BLANK8
DMAINTVECTOR
DMAPORTADDR
FIRSTPORT
FIRSTVECTOR
NBROFPORTS
NBROFTERMINALS
PORTOFFSET
PST
SOCINTVECTOR
SOCPORTADDR
VECTOROFFSET

Globals Written: PST

PDL Description:

```
for all PST entries:
    init  Terminal ID
    init  Port Addr
    init  Interrupt vector
    init  Terminal Mode
    init  Process ID
    init  Process Name
```

C.2.4 InitInterrupts -

Module Number: 1.2

Called From: 1 InitSystem

Modules Called: N/A

Globals Read: DMA
DMACSR
PST

Globals Written: PST

PDL Description:

```
set new interrupt PSW mask = 340 (octal)

save current port interrupt PSW mask
save current port interrupt vector

set current port interrupt mask = new interrupt PSW mask
set current port interrupt vector = PST interrupt vector
```

C.2.5 SOCInterruptServiceRoutine -

Module Number: 1.2.1

Called From: hardware interrupt
(vector addr: 000060
port addr: 777560)

Activated from: 1.2 InitInterrupts

Modules Called: N/A

Globals Read: BACKSPACE
CR
CTRLC
DEL
LF
SPACE
InChar
PST
YES

Globals Written: AbortFlag
InChar
PST

PDL Description:

```
CALL entint to save machine registers

move char from data port to InChar buffer
echo char to console screen

if input char was a carriage return
    echo a line feed to the console screen

casentry -- input char

    case   Control C  (^C)
                set the system abort flag = YES

    case   Delete key
                if this is not the first char
                    decrement the InChar "put" index
                    back-space the console cursor
                    over-write console char with a space
                    back-space the console cursor

    case   default
                increment the InChar "put" index
```

```
                    if InChar "put" index exceeds buffer limit
                    decrement InChar "put" index
        endcase

        CALL retint to restore machine registers
```

C.2.6 T1InterruptServiceRoutine -

Module Number: 1.2.2

Called From: hardware interrupt
(vector addr: 000300
port addr: 776500)

Activated from: 1.2 InitInterrupts

Modules Called: N/A

Globals Read: BACKSPACE
CR
DEL
LF
SPACE
InChar
PST

Globals Written: InChar
PST

PDL Description:

```
CALL entint to save machine registers

move char from data port to InChar buffer
echo char to console screen

if char was a carriage return
    echo a line feed to the console screen

if char was the "delete" key
    if this is not the first char
        decrement the InChar "put" index
        back-space the console cursor
        over-write console char with a space
        back-space the console cursor
else
    increment the InChar "put" index
    if InChar "put" index exceeds buffer limit
        decrement InChar "put" index

CALL retint to restore machine registers
```

C.2.7 T2InterruptServiceRoutine -

Module Number: 1.2.3

Called From: hardware interrupt
(vector addr: 000310
port addr: 776510)

Activated from: 1.2 InitInterrupts

Modules Called: N/A

Globals Read: BACKSPACE
CR
DEL
LF
SPACE
InChar
PST

Globals Written: InChar
PST

PDL Description:

```
CALL entint to save machine registers

move char from data port to InChar buffer
echo char to console screen

if char was a carriage return
    echo a line feed to the console screen

if char was the "delete" key
    if this is not the first char
        decrement the InChar "put" index
        back-space the console cursor
        over-write console char with a space
        back-space the console cursor
else
    increment the InChar "put" index
    if InChar "put" index exceeds buffer limit
        decrement InChar "put" index

CALL retint to restore machine registers
```

C.2.8 T3InterruptServiceRoutine -

Module Number: 1.2.4

Called From: hardware interrupt
(vector addr: 000320
port addr: 776520)

Activated from: 1.2 InitInterrupts

Modules Called: N/A

Globals Read: BACKSPACE
CR
DEL
LF
SPACE
InChar
PST

Globals Written: InChar
PST

PDL Description:

```
CALL entint to save machine registers

move char from data port to InChar buffer
echo char to console screen

if char was a carriage return
    echo a line feed to the console screen

if char was the "delete" key
    if this is not the first char
        decrement the InChar "put" index
        back-space the console cursor
        over-write console char with a space
        back-space the console cursor
else
    increment the InChar "put" index
    if InChar "put" index exceeds buffer limit
        decrement InChar "put" index

CALL retint to restore machine registers
```

C.2.9 T4InterruptServiceRoutine -

Module Number: 1.2.5

Called From: hardware interrupt
(vector addr: 000340
port addr: 776540)

Activated from: 1.2 InitInterrupts

Modules Called: N/A

Globals Read: BACKSPACE
CR
DEL
LF
SPACE
InChar
PST

Globals Written: InChar
PST

PDL Description:

```
CALL entint to save machine registers

move char from data port to InChar buffer
echo char to console screen

if char was a carriage return
    echo a line feed to the console screen

if char was the "delete" key
    if this is not the first char
        decrement the InChar "put" index
        back-space the console cursor
        over-write console char with a space
        back-space the console cursor
else
    increment the InChar "put" index
    if InChar "put" index exceeds buffer limit
        decrement InChar "put" index

CALL retint to restore machine registers
```

C.2.10 T5InterruptServiceRoutine -

Module Number: 1.2.6

Called From: hardware interrupt
(vector addr: 000350
port addr: 776550)

Activated from: 1.2 InitInterrupts

Modules Called: N/A

Globals Read: BACKSPACE
CR
DEL
LF
SPACE
InChar
PST

Globals Written: InChar
PST

PDL Description:

```
CALL entint to save machine registers

move char from data port to InChar buffer
echo char to console screen

if char was a carriage return
    echo a line feed to the console screen

if char was the "delete" key
    if this is not the first char
        decrement the InChar "put" index
        back-space the console cursor
        over-write console char with a space
        back-space the console cursor
else
    increment the InChar "put" index
    if InChar "put" index exceeds buffer limit
        decrement InChar "put" index

CALL retint to restore machine registers
```

C.2.11 T6InterruptServiceRoutine -

Module Number: 1.2.7

Called From: hardware interrupt
(vector addr: 000360
port addr: 776560)

Activated from: 1.2 InitInterrupts

Modules Called: N/A

Globals Read: BACKSPACE
CR
DEL
LF
SPACE
InChar
PST

Globals Written: InChar
PST

PDL Description:

```
CALL entint to save machine registers

move char from data port to InChar buffer
echo char to console screen

if char was a carriage return
    echo a line feed to the console screen

if char was the "delete" key
    if this is not the first char
        decrement the InChar "put" index
        back-space the console cursor
        over-write console char with a space
        back-space the console cursor
else
    increment the InChar "put" index
    if InChar "put" index exceeds buffer limit
        decrement InChar "put" index

CALL retint to restore machine registers
```

C.2.12 T7InterruptServiceRoutine -

Module Number: 1.2.8

Called From: hardware interrupt
(vector addr: 000370
port addr: 776570)

Activated from: 1.2 InitInterrupts

Modules Called: N/A

Globals Read: BACKSPACE
CR
DEL
LF
SPACE
InChar
PST

Globals Written: InChar
PST

PDL Description:

```
CALL entint to save machine registers

move char from data port to InChar buffer
echo char to console screen

if char was a carriage return
    echo a line feed to the console screen

if char was the "delete" key
    if this is not the first char
        decrement the InChar "put" index
        back-space the console cursor
        over-write console char with a space
        back-space the console cursor
else
    increment the InChar "put" index
    if InChar "put" index exceeds buffer limit
        decrement InChar "put" index

CALL retint to restore machine registers
```

C.2.13 DMAInterruptServiceRoutine -

Module Number: 1.2.9

Called From: hardware interrupt
(vector addr: 000124
port addr: 772410)

Activated from: 1.2 InitInterrupts

Modules Called: 1.2.9.1 SetUpForInputDMA

Globals Read: DMABAR
DMACSR
DMADBR
DMAGO
DMAIREQUEST
DMANEX
DMAODIRECTION
DMAOMODE
DMAWCR

DMABusyFlag
DMAwc
NBT
NodeChar

Globals Written: DMABusyFlag
DMAwc
NBT
NodeChar

PDL Description:

```
CALL entint to save machine registers

if a non-existant memory address was referenced
    display a SOC alert
else
    if this is a host input request
        CALL SetUpForInputDMA to service the request
    else
        casentry  --  status of the DMA-busy flag
            case  1  (input, word mode expected)
                fetch the word count from the host
                if enough room in node queue for msg
                    set DMA-busy flag = 2
                    set DMA Base Addr register
                    set DMA Word Count register
                    set DMACSR Output Direction
```

```
                    set DMACSR Output Mode
                    set DMACSR Go bit
                else
                    display SOC alert

            case  2  (input, block mode expected)

            case  3  (output, word mode in progress)

            case  4  (output, block mode in progress)

        endcase

CALL retint to restore machine registers
```

C.2.14 SetUpForInputDMA -

Module Number: 1.2.9.1

Called From: 1.2.9 DMAInterruptServiceRoutine

Modules Called: N/A

Globals Read:
DMA
DMABAR
DMACSR
DMAIMODE
DMAOMODE
DMAWCR

DMABusyFlag
DMAwc
InChar
PST

Globals Written: DMABusyFlag
PST

PDL Description:

```
set DMA-busy flag = input (word mode) expected
set up DMA Base address register

if DMA request is for "word" mode
    set DMA output mode to "word" mode
    set DMA word count = 1
else
    set DMA output mode to "block" mode
    set DMA word count = requested word count
    set DMA-busy flag = input (block mode) expected
```

C.2.15 PerfNormalActivities -

Module Number: 2

Called From: 0 Main

Modules Called: 2.1 SrvInputQueue
2.2 SrvOutputQueue
2.3 SrvNodeQueue

Globals Read: AbortFlag
NBROFBUFFERS
NBROFPORTS
NBT
NO
PST

Globals Written: NBT
PST

PDL Description:

```
display "FEP Activated" console alert
display activation time

while the abort flag remains cleared

    for all Input Queues
        if input chars are queued
           CALL SrvInputQueue
        else
            re-initialize pointers

    for all Output Queues
        if output chars are queued
            CALL SrvOutputQueue
        else
            re-initialize pointers

    for all Node Queues
        if node chars are queued
            CALL SrvNodeQueue
        else
            re-initialize pointers
```

C.2.16 SrvInputQueue -

Module Number: 2.1

Called From: 2 PerfNormalActivities

Modules Called: 2.1.1 EvalSOCInput
2.1.2 MoveMsgtoNodeTTxDMA

Globals Read: CR
DMA
DMATTx
GoDMAFlag
InChar
NBT
NBTCharCount
PST
PSTCharCount
SOC
StartIdx
StopIdx
THTSIZE
TTxDMA

Globals Written: GoDMAFlag
NBT
NBTCharCount
PST
PSTCharCount
StartIdx
StopIdx

PDL Description:

```
while Input Queue chars remain to be evaluated

    if the char is a carriage return or
        the terminal is in character mode

        if input was from SOC
            CALL EvalSOCInput
            set node index for outbound DMA
        else
            if input was from DMA
                set node index for inbound DMA
            else
                set node index for outbound DMA

        if chars are to be moved to DMA
            calculate the message size (chars)
```

```
if the message size is an odd number
    add one for DMA even word transfer

if enough space exists in node queue
    CALL MoveMsgtoNodeTTxDMA
else
    display SOC alert of node saturation
```

C.2.17 EvalSOCInput -

Module Number: 2.1.1

Called From: 2.1 SrvInputQueue

Modules Called: N/A

Globals Read: GoDMAFlag
InChar
NBT
NO
PST
SOC
StartIdx
StopIdx
YES

Globals Written: GoDMAFlag
PST

PDL Description:

```
Assume no DMA output is to occur

if input request is to display the PST
    CALL DispPST
else
    if input request is to display the NBT
        CALL DispNBT
    else
        if input request is to display time
            CALL DispTime
        else
            set DMA-output-is-to-occur flag

if this current input is a display request
    adjust fetch pointer for next input message
```

C.2.18 MoveMsgtoNodeTTxDMA -

Module Number: 2.1.2

Called From: 2.1 SrvInputQueue

Modules Called: 2.1.2.1 BldTransportHeader

Globals Read: InChar
LF
NBT
NodeChar
PSTCharCount
PST
THT
THTSIZE

Globals Written: NBT
NodeChar
PST

PDL Description:

```
CALL BldTransportHeader for current message

for each character to be moved to the node queue

    if the char is part of the Transport Header
        move char from THT to NodeChar
    else
        move char from InChar to NodeChar

if message contains an odd number of chars
    pad message with a trailing line feed char

CALL GatherStats to trap node queue accounting data
```

C.2.19 BldTransportHeader -

Module Number: 2.1.2.1

Called From: 2.1.2 MoveMsgtoNodeTTxDMA

Modules Called: N/A

Globals Read: MAXMSGSEQNBR
ZBTERM
ZERO5

LastMsgSeqNbr
NBT
NBTCharCount
PST

Globals Written: LastMsgSeqNbr
THT

PDL Description:

```
copy process name from PST to THT
copy process ID from PST to THT
copy terminal ID from PST to THT
copy terminal mode from PST to THT

increment the last message sequence number

if last message sequence number exceeds limit
    set last message sequence number = 0

copy ascii value of last message sequence number to THT
set the multi-packet flag in THT = character 'N'
copy ascii value of msg char count to THT
set multi-packet sequence number in THT = '00'
copy originating node name from NBT to THT
copy a zero byte delimiter to THT
```

C.2.20 SrvOutputQueue -

Module Number: 2.2

Called From: 2 PerfNormalActivities

Modules Called: N/A

Globals Read: OutChar
PST

Globals Written: PST

PDL Description:

```
if output port is ready for next character

    move next char from OutChar to output data port
    increment the OutChar "get" pointer

    if the OutChar "get" pointer exceeds high limit
        set OutChar "get" pointer to low limit
```

C.2.21 SrvNodeQueue -

Module Number: 2.3

Called From: 2 PerfNormalActivities

Modules Called: 2.3.1 TTxToDMAOutput
2.3.2 DMAtoTTxOutput

Globals Read: NBROFPORTS
TTxDMA

NBT
NBTCharCount
NodeChar
PST

Globals Written: EndIdx
NBT
NBTCharCount

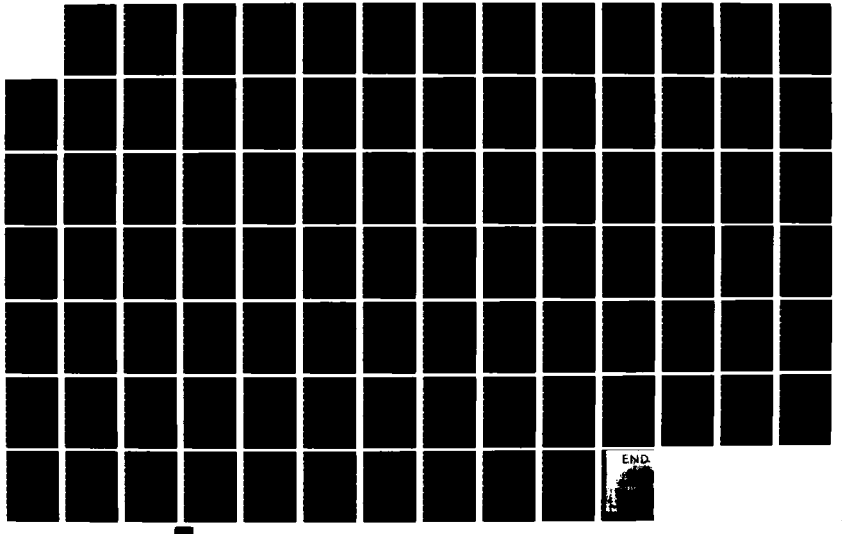
PDL Description:

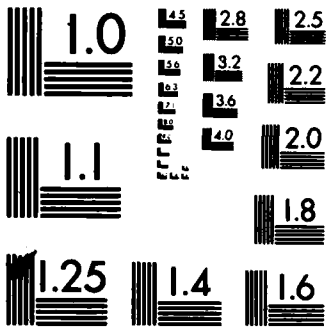
```
While chars remain in the queue
    Do for each entry (port) in the PST
        CALL strcompare to match terminal ID with msg header
        if the msg in the queue is for this terminal user
            get msg char count from Transport header
            convert this ascii count to an integer
            set EndIdx = NodeChar index of the last char
            if the queue being serviced is TTx-to-DMA
                CALL TTxtoDMAOutput to request DMA transfer
            else
                CALL DMAtoTTxOutput to move msg to OutChar

    if strcompare could find no matching terminal ID
        display an error alert upon the SOC terminal screen
        display the msg contents on the SOC terminal screen
        flush entire node buffer by resetting the 'get' ptr
```

AD-A138 152 DEVELOPMENT OF A COMMUNICATIONS FRONT END PROCESSOR (FEP) FOR THE VAX-11/.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. A F MASTY 3/3

UNCLASSIFIED DEC 83 AFIT/GCS/EE/83D-13 F/G 17/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

C.2.22 TTxtoDMAOutput -

Module Number: 2.3.1

Called From: 2.3 SrvNodeQueue

Modules Called: N/A

Globals Read: DMACSR
DMAREADY
DMAODIRECTION
DMAOMODE
TTxDMA

DMABusyFlag
EndIdx
NBT
NodeChar

Globals Written: DMACSR
DMADBR
DMAODIRECTION
DMAOMODE

DMABusyFlag
NBT

PDL Description:

```
if 'word' mode input is expected and DMA is not busy
    set DMA Base Addr register = block word count
    set DMA Output mode = 'word' mode
    set DMA Output direction = LSI is transmitter
    set DMA-busy flag = Output 'word' mode in progress
else
    display an error alert msg upon the SOC terminal screen

CALL GatherStats to trap node queue accounting data
```

C.2.23 DMAtoTTxOutput -

Module Number: 2.3.2

Called From: 2.3 SrvNodeQueue

Modules Called: N/A

Globals Read: DMATTx

EndIdx
NBT
NodeChar
PST

Globals Written: NBT
OutChar
PST

PDL Description:

```
CALL GatherStats to trap node queue accounting data

while NodeChar characters remain for this message
    if OutChar 'put' index < high limit
        copy char from NodeChar to OutChar
        increment NodeChar pointer to next char
        increment OutChar pointer for next char
```

C.2.24 TermSystem -

Module Number: 3

Called From: 0 Main

Modules Called: N/A

Globals Read: NBROFPORTS

PST

Globals Written: N/A

PDL Description:

```
Do for all PST entries
    restore original interrupt PWS mask
    restore original interrupt vector address

display a SOC alert that the FEP system has been aborted

CALL DispTime to display the time of abort

CALL DispElapsedTime to display the duration of FEP processing

CALL fclose to close the accounting file
```

APPENDIX D

LSI FEP SOURCE CODE LISTINGS

These source listings represent the latest versions of the LSI FEP programs LFEPLO.C and LFEPHI.C. Modules forming LFEPLO.C are identified by module numbers containing purely numerical terms. Modules forming LFEPHI.C are identified by prefixing the numerical part with the 'X' character representing extended memory mapping.

Table of Contents for Appendix D

D.1 LFEPLO.C Program modules

```
/*****************************************************************
*                                                                *
*    TITLE:                LSI FEP Low Memory 'C' Program        *
*                                                                *
*    FILENAME:             LFEPLO.C                              *
*                                                                *
*    DATE:                 3 Nov 83                              *
*    VERSION:              A1                                    *
*    OWNER:                Capt Allan F. Masty                   *
*    COMPUTER SYSTEM:      LSI-11/23                             *
*    OPERATING SYSTEM:     RT11XM                                *
*    LANGUAGE:             Telecon 'C'                           *
*                                                                *
*    CONTENTS:             0          Main                       *
*                          1          InitSystem                 *
*                          1.1        InitPST                    *
*                          1.2        InitInterrupts             *
*                          1.2.1      SOCInterruptServiceRoutine *
*                          1.2.2      T1InterruptServiceRoutine  *
*                          1.2.3      T2InterruptServiceRoutine  *
*                          1.2.4      T3InterruptServiceRoutine  *
*                          1.2.5      T4InterruptServiceRoutine  *
*                          1.2.6      T5InterruptServiceRoutine  *
*                          1.2.7      T6InterruptServiceRoutine  *
*                          1.2.8      T7InterruptServiceRoutine  *
*                          1.2.9      DMAInterruptServiceRoutine *
*                          1.2.9.1    SetUpForInputDMA           *
*                          2          PerfNormalActivities       *
*                          2.1        SrvInputQueue              *
*                          2.1.1      EvalSOCInput               *
*                          2.1.2      MoveMsgtoNodeTTxDMA        *
*                          2.1.2.1    BldTransportHeader         *
*                          2.2        SrvOutputQueue             *
*                          2.3        SrvNodeQueue               *
*                          2.3.1      TTxtoDMAOutput             *
*                          2.3.2      DMAtoTTxOutput             *
*                          3          TermSystem                 *
*                                                                *
*    FUNCTION:             Performs as a Communications Front End *
*                          Processor (FEP) for a DEC VAX-11/780.  *
*                          Functions as a Terminal Concentrator in *
*                          assembling and routing the keyboard and *
*                          screen traffic for 8 (expandable to 16) *
*                          DEC VT-100 terminals.                  *
*                                                                *
*****************************************************************/
```

```
/* ..................  GLOBALS  ..................... */

#define BACKSPACE          '\010' /* Back-space char code      */
#define CR                 '\015' /* Carriage Return char      */
#define CTRLC              '\003' /* ^C ( control-C ) char     */
#define DEL                '\177' /* "DELETE" key char code    */
#define FIRSTPORT         0176500 /* First LSI-11 port addr    */
#define FIRSTVECTOR       0000300 /* First int. vector addr    */
#define LF                 '\012' /* Line Feed character       */
#define MAXMSGSEQNBR      0007777 /* Largest message number    */
#define NO                      0 /* BOOLEAN variable          */
#define PORTOFFSET              8 /* 8 word port seperations   */
#define SOCPORTADDR       0177560 /* I/O port addr for SOC     */
#define SOCINTVECTOR      0000060 /* SOC Interrupt vector addr*/
#define SPACE              '\040' /* Space character code      */
#define TERMBUFFERSIZE        200 /* Terminal buffer size      */
#define VECTOROFFSET            8 /* Interrupt vector spacing */
#define YES                     1 /* BOOLEAN variable          */

extern  char    InChar   [] ,    /* Terminal  Input Buffers  */
                OutChar  [] ,    /* Terminal Output Buffers  */
                NodeChar [] ;    /* Node-to-Node  Buffers    */

#include        "lfepio.h"       /*   Standard IO routines   */

extern  int     fopen () ;

int     AbortFlag = NO ,         /* BOOLEAN for aborting FEP */
        DMABusyFlag ,            /* Flag set when the DMA
                                        interface is busy :

                                 0 = Not Busy
                                 1 = DMA word  input expected
                                 2 = DMA block input expected
                                 3 = DMA word  output pending
                                 4 = DMA block output pending   */

        DMAwc ,                  /* DMA word count (block)       */
        EndIdx ,                 /* Used in servicing NodeChar   */
        FileOpenFlag ,           /* Status of file LSIFEP.DAT    */
        GoDMAFlag ,              /* BOOLEAN to send DMA output   */
        StartIdx ,               /* Used in servicing InChar     */
        StopIdx ,                /* Used in servicing InChar     */
        LastMsgSeqNbr ,          /* Increments for each new msg */
        NBTCharCount ,           /* Count of chars in NBT msg    */
        PSTCharCount ;           /* Count of chars in PST msg    */

FILE    *fp ,
        *fopen () ;
```

```
/*  .  Port Status ( PS ) Table defines & declarations  . . */

#define NBROFPORTS       9        /* NBROFTERMINALS +SOC +DMA */
#define NBROFTERMINALS   7        /* # of VT-100 terminals    */
#define          SOC     0        /* System Operator's Console*/
#define          T1      1        /* VT-100 unit #1           */
#define          T2      2        /* VT-100 unit #2           */
#define          T3      3        /* VT-100 unit #3           */
#define          T4      4        /* VT-100 unit #4           */
#define          T5      5        /* VT-100 unit #5           */
#define          T6      6        /* VT-100 unit #6           */
#define          T7      7        /* VT-100 unit #7           */
#define          DMA     8        /* Direct Memory Access     */

struct  PortStatusDataRecord
        {
        char    TID [4] ;         /* VT-100 Terminal ID       */
        char    TermMode ;        /* line or character mode   */

        int     InLowIdx   ;      /* Input buffer pointers    */
        int     InPutIdx   ;
        int     InGetIdx   ;
        int     InHighIdx  ;

        int     OutLowIdx  ;      /* Output buffer pointers   */
        int     OutPutIdx  ;
        int     OutGetIdx  ;
        int     OutHighIdx ;

        int     *RcvStatAddr ;    /* receive port status addr */
        int     *RcvDataAddr ;    /* receive port data addr   */
        int     *TxmStatAddr ;    /* transmit port status addr*/
        int     *TxmDataAddr ;    /* transmit port data addr  */

        int     *IntVectAddr ;    /* receive port int. addr   */
        int     IntRoutineAddr ; /* Interrupt service routine*/

        int     StorIntVect  ;
        int     StorPSW      ;
        }
PST [ NBROFPORTS ]  ;        /* I/O Port Status Table ( PST ) */
```

```
/*  .  index  DEFINES for node - to - node buffers  . . .   */

#define          TTxDMA  0        /* TTx to DMA buffer        */
#define          DMATTx  1        /* DMA to TTx buffer        */

/*   . . .  misc defines for Node Buffer table   . . . . .  */

#define NBROFBUFFERS          2 /* Number of Node buffers   */
#define NODEBUFFERSIZE     2000 /* # of chars in each buffer*/

/*    . . . Node Buffer Table declaration   . . . . . . . .  */

struct  NodeBufferRecord
        {
        char    OrgNode [4] ;
        int     LowIdx  ;
        int     PutIdx  ;
        int     GetIdx  ;
        int     HighIdx ;
        }
NBT [ NBROFBUFFERS ] ;        /* Node Buffer Table ( NBT ) */


struct  MsgTransportLayerHeader
        {
        char    TID [3] ;
        char    Mode ;
        char    MsgSeqNbr [4] ;
        char    MsgCharCnt [4] ;
        char    OrgNode [3] ;
        }
THT ;                         /* Transport Header Table ( THT ) */

#define          THTSIZE sizeof ( THT )
```

```
/*      . . . .   Defines for DMA control   . . . .          */

#define DMAINTVECTOR    0000124 /* DMA interrupt vector      */
#define DMAPORTADDR     0172410 /* DMA start port address    */
#define DMAWCR          0172410 /* DMA Word Count Register   */
#define DMABAR          0172412 /* DMA Bus Address Register  */
#define DMACSR          0172414 /* DMA Control/Status Reg    */
#define DMADBR          0172416 /* DMA Data Buffer Register  */

/* ... the following define bit settings in DMACSR  .... */

#define DMAGO           0000001 /* bit  0 */
#define DMAOMODE        0000002 /* bit  1 */
#define DMAODIRECTION   0000004 /* bit  2 */
#define DMAOREQUEST     0000010 /* bit  3 */
                                /* bit  4 not used */
                                /* bit  5 not used */
#define DMAIE           0000100 /* bit  6 */
#define DMAREADY        0000200 /* bit  7 */
#define DMACYCLE        0000400 /* bit  8 */
#define DMAIMODE        0001000 /* bit  9 */
#define DMAIDIRECTION   0002000 /* bit 10 */
#define DMAIREQUEST     0004000 /* bit 11 */
                                /* bit 12 not used */
                                /* bit 13 not used */
#define DMANEX          0040000 /* bit 14 */
#define DMAERROR        0100000 /* bit 15 */
```

D.1.1 Main

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       0  -  Main                      *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Top level module for "LFEPLO.C" *
*    INPUTS:                     NONE                            *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      NONE                            *
*    GLOBAL VARIABLES CHANGED:   NONE                            *
*    GLOBAL TABLES USED:         NONE                            *
*    GLOBAL TABLES CHANGED:      NONE                            *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             1  -  InitSystem                *
*                                2  -  PerfNormalAcivities       *
*                                3  - TermSystem                 *
*    CALLING MODULES:            NONE                            *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

Main ()
    {
    InitSystem ();              /* First-time processing     */
    PerfNormalActivities ();    /* Synchronous processing    */
    TermSystem ();              /* Clean-up routines         */
    exit (0);                   /* return to RT-11 monitor   */
    }
```

D.1.2 InitSystem

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       1  -  InitSystem                *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Initializes data base and opens *
*                                accounting file "LSIFEP.DAT".   *
*    INPUTS:                     NONE                            *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      NONE                            *
*    GLOBAL VARIABLES CHANGED:   DMABusyFlag                     *
*                                FileOpenFlag                    *
*    GLOBAL TABLES USED:         NBT                             *
*                                PST                             *
*    GLOBAL TABLES CHANGED:      NBT                             *
*                                PST                             *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             1.1  -  InitPST                 *
*                                1.2  -  InitInterrupts          *
*    CALLING MODULES:            0  -  Main                      *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

InitSystem ()
    {
    int     i;                      /* Loop control variable */

    DMABusyFlag = 0 ;           /* Input word [mode] expected */
    FileOpenFlag = YES ;

    if ( ( fp = fopen ( "LSIFEP.DAT", "w" ) ) == NULL )
        {
        FileOpenFlag =  NO ;
        printf ("\n LSIFEP.DAT cannot be opened. \n" ) ;
        }

    for ( i = 0; i < NBROFPORTS; i++ )
        {
        PST [i].InLowIdx    = i * TERMBUFFERSIZE ;
        PST [i].InPutIdx    = PST [i].InLowIdx   ;
        PST [i].InGetIdx    = PST [i].InLowIdx   ;
        PST [i].InHighIdx   = PST [i].InLowIdx+TERMBUFFERSIZE-1;
        PST [i].OutLowIdx   = i * TERMBUFFERSIZE ;
        PST [i].OutPutIdx   = PST [i].OutLowIdx  ;
```

```
        PST [i].OutGetIdx  = PST [i].OutLowIdx  ;
        PST [i].OutHighIdx = PST [i].OutLowIdx+TERMBUFFERSIZE-1;
        }

    for ( i = 0; i < NBROFBUFFERS; i++ )
        {
        NBT [i].LowIdx     = i * NODEBUFFERSIZE ;
        NBT [i].PutIdx     = NBT [i].LowIdx ;
        NBT [i].GetIdx     = NBT [i].LowIdx ;
        NBT [i].HighIdx    = NBT [i].LowIdx+NODEBUFFERSIZE-1;
        }

    strcpy ( NBT [ TTxDMA ].OrgNode, "TTx" ) ;
    strcpy ( NBT [ DMATTx ].OrgNode, "DMA" ) ;

    PST [SOC].IntRoutineAddr = SOCInterruptServiceRoutine ;
    PST [ T1].IntRoutineAddr =  T1InterruptServiceRoutine ;
    PST [ T2].IntRoutineAddr =  T2InterruptServiceRoutine ;
    PST [ T3].IntRoutineAddr =  T3InterruptServiceRoutine ;
    PST [ T4].IntRoutineAddr =  T4InterruptServiceRoutine ;
    PST [ T5].IntRoutineAddr =  T5InterruptServiceRoutine ;
    PST [ T6].IntRoutineAddr =  T6InterruptServiceRoutine ;
    PST [ T7].IntRoutineAddr =  T7InterruptServiceRoutine ;
    PST [DMA].IntRoutineAddr = DMAInterruptServiceRoutine ;

    InitPST () ;          /* Initialize Port Status Table */

    InitInterrupts () ;  /* Turn on all interrupts */

    return ;
    }
```

D.1.3 InitPST

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       1.1  -  InitPST                 *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Completes PST initialization    *
*    INPUTS:                     NONE                            *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      NONE                            *
*    GLOBAL VARIABLES CHANGED:   NONE                            *
*    GLOBAL TABLES USED:         PST                             *
*    GLOBAL TABLES CHANGED:      PST                             *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             NONE                            *
*    CALLING MODULES:            1  -  InitSystem                *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

InitPST ( )

    {

    int         i ;             /* index into PST  */

    strcpy ( PST [SOC].TID, "SOC" ) ;
    strcpy ( PST [ T1].TID, "T.1" ) ;
    strcpy ( PST [ T2].TID, "T.2" ) ;
    strcpy ( PST [ T3].TID, "T.3" ) ;
    strcpy ( PST [ T4].TID, "T.4" ) ;
    strcpy ( PST [ T5].TID, "T.5" ) ;
    strcpy ( PST [ T6].TID, "T.6" ) ;
    strcpy ( PST [ T7].TID, "T.7" ) ;
    strcpy ( PST [DMA].TID, "DMA" ) ;

    for ( i = 1; i <= 3; i++ )      /*   VT-100's only  */
        {
        PST [i].RcvStatAddr  = FIRSTPORT   +
                               ( (i-1) * PORTOFFSET )   ;
        PST [i].IntVectAddr  = FIRSTVECTOR +
                               ( (i-1) * VECTOROFFSET ) ;
        }

    for ( i = 4; i <= NBROFTERMINALS; i++ )
        {
```

```
        PST [i].RcvStatAddr   = FIRSTPORT    +
                                ( i * PORTOFFSET )    ;
        PST [i].IntVectAddr   = FIRSTVECTOR +
                                ( i * VECTOROFFSET ) ;
        }

PST [SOC].RcvStatAddr     = SOCPORTADDR  ;
PST [SOC].IntVectAddr     = SOCINTVECTOR ;
PST [DMA].RcvStatAddr     = DMAPORTADDR  ;
PST [DMA].IntVectAddr     = DMAINTVECTOR ;

for ( i = 0 ; i < NBROFPORTS; i++ )
        {                              /*  all table entries  */
        PST [i].TermMode      = 'L'          ;
        PST [i].RcvDataAddr   = PST [i].RcvStatAddr + 2 ;
        PST [i].TxmStatAddr   = PST [i].RcvStatAddr + 4 ;
        PST [i].TxmDataAddr   = PST [i].RcvStatAddr + 6 ;
        }

return ;
}
```

D.1.4 InitInterrupts

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:      1.2  -  InitInterrupts          *
*                                                                *
*    DATE:                      3 Nov 83                        *
*    VERSION:                   A1                              *
*    FUNCTION:                  Enables interrupts for all PST  *
*                               entries                         *
*    INPUTS:                    NONE                            *
*    OUTPUTS:                   NONE                            *
*    GLOBAL VARIABLES USED:     NONE                            *
*    GLOBAL VARIABLES CHANGED:  NONE                            *
*    GLOBAL TABLES USED:        PST                             *
*    GLOBAL TABLES CHANGED:     PST                             *
*    FILES READ:                NONE                            *
*    FILES WRITTEN:             NONE                            *
*    MODULES CALLED:            NONE                            *
*    CALLING MODULES:           1  -  InitSystem                *
*                                                                *
*    AUTHOR:                    Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                   VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

InitInterrupts ( )

    {
    int       i ,          /* index into PST */
              PSW ,        /* Processor Status Word */
              *temp ;      /* temporary pointer */

    PSW          = 0000340 ;      /* Interrupt vector PSW */

    for ( i = 0; i < NBROFPORTS; i++ )
        {
        temp     = PST [i].IntVectAddr  ;

        PST [i].StorIntVect    = *( temp )     ;
        PST [i].StorPSW        = *(temp+1)     ;
        *PST [i].IntVectAddr   = PST [i].IntRoutineAddr ;
        *(temp+1)              = PSW           ;

        if ( i < DMA )                /* enable interrupts */
            *PST [i].RcvStatAddr = 0100 ;
        else
            *DMACSR           = 0100 ;
        }
return  ;
}
```

D.1.5 PerfNormalActivities

```
/*****************************************************************
*                                                                *
*     MODULE NUMBER / NAME:       2  -  PerfNormalActivities     *
*                                                                *
*     DATE:                       3 Nov 83                       *
*     VERSION:                    A1                             *
*     FUNCTION:                   Loops and scans node queues    *
*                                 InChar, OutChar, and NodeChar  *
*                                 for message traffic to move.   *
*     INPUTS:                     NONE                           *
*     OUTPUTS:                    NONE                           *
*     GLOBAL VARIABLES USED:      AbortFlag                      *
*     GLOBAL VARIABLES CHANGED:   NONE                           *
*     GLOBAL TABLES USED:         NBT                            *
*                                 PST                            *
*     GLOBAL TABLES CHANGED:      NBT                            *
*                                 PST                            *
*     FILES READ:                 NONE                           *
*     FILES WRITTEN:              NONE                           *
*     MODULES CALLED:             X.4  -  DispTime               *
*                                 2.1  -  SrvInputQueue          *
*                                 2.2  -  SrvOutputQueue         *
*                                 2.3  -  SrvNodeQueue           *
*     CALLING MODULES:            0  -  Main                     *
*                                                                *
*     AUTHOR:                     Capt Allan F. Masty, GCS-83D   *
*     HISTORY:                    VSN A1 -  3 November 1983      *
*                                                                *
*****************************************************************/

PerfNormalActivities ()

    {

    int         i ;                 /* index into PST or NBT */

    printf ("\n++++++++ FEP Activated at ") ;

    DispTime () ;

    while ( AbortFlag == NO )
        {
        for ( i = NBROFPORTS - 1; i >= 0; i-- )
            {                       /* Input queue scans */
            if ( PST [i].InPutIdx != PST [i].InGetIdx )
                SrvInputQueue ( i ) ;
            else
                {
                PST[i].InPutIdx = PST[i].InLowIdx ;
```

```
                PST[i].InGetIdx = PST[i].InLowIdx ;
                }
            }

        for ( i = NBROFPORTS - 1; i >= 0; i-- )
            {                       /* Output queue scans */
            if ( PST [i].OutPutIdx != PST [i].OutGetIdx )
                SrvOutputQueue ( i ) ;
            else
                {
                PST[i].OutPutIdx = PST[i].OutLowIdx ;
                PST[i].OutGetIdx = PST[i].OutLowIdx ;
                }
            }

        for ( i = NBROFBUFFERS -1; i >= 0; i-- )
            {                       /* Node buffer scans  */
            if ( NBT [i].PutIdx != NBT [i].GetIdx )
                SrvNodeQueue ( i ) ;
            else
                {
                NBT [i].PutIdx = NBT [i].LowIdx ;
                NBT [i].GetIdx = NBT [i].LowIdx ;
                }
            }
        }
    return ;
    }
```

D.1.6 SrvInputQueue

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:        2.1  -  SrvInputQueue          *
*                                                                *
*    DATE:                        3 Nov 83                       *
*    VERSION:                     A1                             *
*    FUNCTION:                    Moves character data from queue *
*                                 InChar to queue NodeChar.      *
*    INPUTS:                      i = index into PST             *
*    OUTPUTS:                     NONE                           *
*    GLOBAL VARIABLES USED:       GoDMAFlag                      *
*                                 NBTCharCount                   *
*                                 PSTCharCount                   *
*                                 StartIdx                       *
*                                 StopIdx                        *
*    GLOBAL VARIABLES CHANGED:    GoDMAFlag                      *
*                                 NBTCharCount                   *
*                                 PSTCharCount                   *
*                                 StartIdx                       *
*                                 StopIdx                        *
*    GLOBAL TABLES USED:          Inchar                         *
*                                 NBT                            *
*                                 PST                            *
*    GLOBAL TABLES CHANGED:       NBT                            *
*                                 PST                            *
*    FILES READ:                  NONE                           *
*    FILES WRITTEN:               NONE                           *
*    MODULES CALLED:              2.1.1  -  EvalSOCInput         *
*                                 2.1.2  -  MoveMsgtoNodeTTxDMA  *
*    CALLING MODULES:             2  -  PerfNormalActivities     *
*                                                                *
*    AUTHOR:                      Capt Allan F. Masty, GCS-83D   *
*    HISTORY:                     VSN A1 -  3 November 1983      *
*                                                                *
*****************************************************************/

SrvInputQueue ( i )

    int     i ;                 /* index into the PST */
    {

    StopIdx = StartIdx = PST [i].InGetIdx ;

    while (  StopIdx < PST [i].InPutIdx )
        {
        if ( ( InChar [StopIdx++] == CR)   ||
                        (PST [i].TermMode == 'C') )
            {
            if ( i == SOC )
```

```
                EvalSOCInput () ;

            if ( GoDMAFlag++ )  /* chars to be moved to DMA */
                {
                PSTCharCount = StopIdx - StartIdx + THTSIZE ;
                NBTCharCount = PSTCharCount ;

                if ( NBTCharCount & 0000001 )
                    ++NBTCharCount ;

                if (( NBT [TTxDMA].PutIdx + NBTCharCount )
                                    <=   NBT [TTxDMA].HighIdx )
                    MoveMsgtoNodeTTxDMA ( i ) ;
                else
                    printf ("\n Node %s saturated.\n",
                                NBT [TTxDMA].OrgNode ) ;
                }
            }                   /* end of "if" processing */
        }               /* end of "while" loop */
    return ;
    }
```

D.1.7 EvalSOCInput

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:      2.1.1  -  EvalSOCInput           *
*                                                                *
*    DATE:                      3 Nov 83                         *
*    VERSION:                   A1                               *
*    FUNCTION:                  Displays system status data on   *
*                               System Operator's Console (SOC)  *
*                               request.                         *
*    INPUTS:                    NONE                             *
*    OUTPUTS:                   NONE                             *
*    GLOBAL VARIABLES USED:     GoDMAFlag                        *
*                               StartIdx                         *
*                               StopIdx                          *
*    GLOBAL VARIABLES CHANGED:  GoDMAFlag                        *
*    GLOBAL TABLES USED:        InChar                           *
*                               NBT                              *
*                               PST                              *
*    GLOBAL TABLES CHANGED:     PST                              *
*    FILES READ:                NONE                             *
*    FILES WRITTEN:             NONE                             *
*    MODULES CALLED:            X.1  -  DispPST                  *
*                               X.2  -  DispNBT                  *
*                               X.4  -  DispTime                 *
*    CALLING MODULES:           2.1  -  SrvInputQueue            *
*                                                                *
*    AUTHOR:                    Capt Allan F. Masty, GCS-83D     *
*    HISTORY:                   VSN A1 -  3 November 1983        *
*                                                                *
*****************************************************************/

EvalSOCInput ()
    {
    char            *p ;

    int z;

    GoDMAFlag = NO ;        /* assume local SOC processing */

    p = &( InChar [StartIdx] ) ;

    if ( strcompare ("pst", p, 3) == 3 )
        DispPST ( &PST ) ;
    else
        if ( strcompare ("nbt", p, 3) == 3 )
            DispNBT ( &NBT ) ;
        else
            if ( strcompare ("time", p, 4) == 4 )
                {
```

```
                printf ("\nTime = ") ;
                DispTime () ;
                printf ("\n") ;
                }
            else        /* set up for DMA transfer */
                GoDMAFlag = YES ;

    if ( GoDMAFlag == NO )
        PST [SOC].InGetIdx = StopIdx ;

    return ;
    }
```

D.1.8 MoveMsgtoNodeTTxDMA

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:        2.1.2  -  MoveMsgtoNodeTTxDMA  *
*                                                                *
*    DATE:                        3 Nov 83                       *
*    VERSION:                     A1                             *
*    FUNCTION:                    Moves chars from queue InChar  *
*                                 to queue NodeChar              *
*    INPUTS:                      i = index into PST             *
*    OUTPUTS:                     NONE                           *
*    GLOBAL VARIABLES USED:       PSTCharCount                   *
*    GLOBAL VARIABLES CHANGED:    NONE                           *
*    GLOBAL TABLES USED:          InChar                         *
*                                 NBT                            *
*                                 PST                            *
*                                 THT                            *
*    GLOBAL TABLES CHANGED:       NBT                            *
*                                 NodeChar                       *
*                                 PST                            *
*    FILES READ:                  NONE                           *
*    FILES WRITTEN:               NONE                           *
*    MODULES CALLED:              2.1.2.1  -  BldTransportHeader *
*                                 X.7  -  GatherStats            *
*    CALLING MODULES:             2.1  -  SrvInputQueue          *
*                                                                *
*    AUTHOR:                      Capt Allan F. Masty, GCS-83D   *
*    HISTORY:                     VSN A1 -  3 November 1983      *
*                                                                *
*****************************************************************/

MoveMsgtoNodeTTxDMA ( i )

int     i ;             /* index into PST */

    {
    char        *CharPtr ;
    int         j ;

    BldTransportHeader ( i ) ;

    for ( CharPtr = THT, j = 0; j < PSTCharCount; j++ )
        {
        if ( j < THTSIZE )
            NodeChar[NBT[TTxDMA].PutIdx++] = *(CharPtr++);
        else
            NodeChar[NBT[TTxDMA].PutIdx++] = InChar[PST[i].InGetIdx+
        }

    if ( PSTCharCount & 0000001 )    /* if odd # of characters  */
```

```
        NodeChar [ NBT [TTxDMA].PutIdx++ ] = LF ;

    GatherStats ( &THT, 1 ) ;

    return ;
    }
```

D.1.9 BldTransportHeader

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       2.1.2.1  -  BldTransportHeader  *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Writes into the 15 character    *
*                                skeleton "THT" to create the    *
*                                Message Transport Layer Header. *
*    INPUTS:                     i = index into PST              *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      NBTCharCount                    *
*    GLOBAL VARIABLES CHANGED:   LastMsgSeqNbr                   *
*    GLOBAL TABLES USED:         PST                             *
*    GLOBAL TABLES CHANGED:      THT                             *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             NONE                            *
*    CALLING MODULES:            2.1.2  -  MoveMsgtoNodeTTxDMA   *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

BldTransportHeader ( i )

int     i ;             /* PST index */

    {

    strcopy  ( THT.TID,           PST [i].TID,  3 ) ;
               THT.Mode         = PST [i].TermMode  ;

    if (++LastMsgSeqNbr > MAXMSGSEQNBR )
        LastMsgSeqNbr = 0 ;

    intascii ( THT.MsgSeqNbr,     LastMsgSeqNbr,   4 ) ;
    intascii ( THT.MsgCharCnt,    NBTCharCount,    4 ) ;
    strcopy  ( THT.OrgNode,       "TTx", 3 ) ;

    return ;
    }
```

D.1.10 SrvOutputQueue

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       2.2  -  SrvOutputQueue          *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Moves next screen character to  *
*                                Transmit Data Buffer (XBUF) if  *
*                                Transmit Control & Status       *
*                                Register (XCSR) indicates that  *
*                                the buffer is ready for it.     *
*    INPUTS:                     i = index into PST              *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      NONE                            *
*    GLOBAL VARIABLES CHANGED:   NONE                            *
*    GLOBAL TABLES USED:         PST                             *
*    GLOBAL TABLES CHANGED:      OutChar                         *
*                                PST                             *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             NONE                            *
*    CALLING MODULES:            2  - PerfNormalActivities       *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

SrvOutputQueue ( i )

int     i ;                     /* index into the PST */

    {
    printf ("\n ???????? SrvOutputQueue ( %d ) ", i ) ;

    if ( *PST [i].TxmStatAddr & 0200 )
        {
        *PST [i].TxmDataAddr = OutChar [PST[i].OutGetIdx] ;
        if ( ++PST [i].OutGetIdx > PST [i].OutHighIdx )
            PST [i].OutGetIdx = PST [i].OutLowIdx ;
        }
    return ;
    }
```

D.1.11 SrvNodeQueue

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:        2.3  -  SrvNodeQueue           *
*                                                                *
*    DATE:                        3 Nov 83                       *
*    VERSION:                     A1                             *
*    FUNCTION:                    Controls movement of chars from *
*                                 NodeChar to the DMA interface  *
*                                 or to OutChar.                 *
*    INPUTS:                      n = index into NBT             *
*    OUTPUTS:                     NONE                           *
*    GLOBAL VARIABLES USED:       NBTCharCount                   *
*    GLOBAL VARIABLES CHANGED:    EndIDX                         *
*                                 NBTCharCount                   *
*    GLOBAL TABLES USED:          NBT                            *
*                                 NodeChar                       *
*                                 PST                            *
*                                 THT                            *
*    GLOBAL TABLES CHANGED:       NBT                            *
*    FILES READ:                  NONE                           *
*    FILES WRITTEN:               NONE                           *
*    MODULES CALLED:              2.3.1  -  TTxtoDMAOutput       *
*                                 2.3.2  -  DMAtoTTxOutput       *
*    CALLING MODULES:             2  -  PerfNormalActivities     *
*                                                                *
*    AUTHOR:                      Capt Allan F. Nasty, GCS-83D   *
*    HISTORY:                     VSN A1 -  3 November 1983      *
*                                                                *
*****************************************************************/

SrvNodeQueue ( n )

    int     n ;                 /* index into the NBT*/

    {
    int     i ,
            p ;

    char   *c ;

    while ( NBT [n].GetIdx != NBT [n].PutIdx )
        {
        i = NBROFPORTS ;

        for ( p = 0; p < NBROFPORTS; p++ )
            {
            if ( strcompare (PST[p].TID,
                    &NodeChar[NBT[n].GetIdx], 3) == 3 )
                {
```

```
                    i = p ;
                    p = NBT[n].GetIdx + &(THT.MsgCharCnt) - &(THT);

                    c = &( NodeChar [p] ) ;

                    NBTCharCount = asciiint ( c, 4 ) ;

                    EndIdx  = NBT [ n ].GetIdx + NBTCharCount ;

                    if ( n == TTxDMA )
                        TTxtoDMAOutput () ;
                    else
                        DMAtoTTxOutput ( i ) ;
                    }                                   /* end 'if' */
                }                                     /* end 'for' */

            if ( i == NBROFPORTS )
                {
                printf ("\n Invalid TID = ") ;
                while ( NBT [ n ].GetIdx != NBT [n].PutIdx )
                    printf ("%c", NodeChar [NBT[n].GetIdx++] );
                printf ("\n\n") ;
                }
            }                           /* end 'while' */
        return ;
        }
```

D.1.12 TTxtoDMAOutput

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       2.3.1  -  TTxtoDMAOutput        *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Controls movement of messages   *
*                                from NodeChar to DMA interface  *
*    INPUTS:                     NONE                            *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      DMABusyFlag                     *
*                                EndIDX                          *
*    GLOBAL VARIABLES CHANGED:   DMABusyFlag                     *
*    GLOBAL TABLES USED:         NBT                             *
*                                NodeChar                        *
*    GLOBAL TABLES CHANGED:      NBT                             *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             X.7  -  GatherStats             *
*    CALLING MODULES:            2.3  -  SrvNodeQueue            *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

TTxtoDMAOutput ()
    {
    if ( ( DMABusyFlag == 0 )   &&   ( *DMACSR & DMAREADY) )
        {
        *DMADBR      = -((EndIdx - NBT [TTxDMA].GetIdx) / 2 ) ;
        *DMACSR     &= ~(DMAOMODE) ;
        *DMACSR     |=   DMAODIRECTION ;
        DMABusyFlag  = 3 ;       /* Output word in progress */
        }
    else
        {
        printf ("\nTTxtoDMAOutput = DMA busy --") ;
        printf ("  DMACSR = %o", *DMACSR ) ;
        printf ("  DMABusyFlag = %d", DMABusyFlag) ;
        }

    GatherStats ( &NodeChar [ NBT[TTxDMA].GetIdx ], 2 ) ;

    while ( NBT [TTxDMA].GetIdx != EndIdx )
        printf ("%c", NodeChar [ NBT [TTxDMA].GetIdx++ ] ) ;

    return ;
    }
```

D.1.13 DMAtoTTxOutput

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       2.3.2  -  DMAtoTTxOutput        *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Controls movement of messages   *
*                                from NodeChar to OutChar        *
*    INPUTS:                     i = index into PST              *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      EndIDX                          *
*    GLOBAL VARIABLES CHANGED:   NONE                            *
*    GLOBAL TABLES USED:         NBT                             *
*                                NodeChar                        *
*                                PST                             *
*                                THT                             *
*    GLOBAL TABLES CHANGED:      NBT                             *
*                                PST                             *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             X.7  -  GatherStats             *
*    CALLING MODULES:            2.3  -  SrvNodeQueue            *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

DMAtoTTxOutput ( i )
    int      i ;                    /* index into PST */
    {
    int      p ;

    GatherStats ( &NodeChar [ NBT[DMATTx].GetIdx ], 2 ) ;

    p = NBT [DMATTx].GetIdx + &(THT.Mode) - &(THT) ;
    PST [i].TermMode = NodeChar [p] ;  /* update LSI database */

    printf ("\nDMAtoTTxTransfer = ") ;

    while ( ( NBT [DMATTx].GetIdx != EndIdx )  &&
            ( PST [i].OutPutIdx < PST [i].OutHighIdx )  )
        {
        printf ("%c", NodeChar [ NBT [DMATTx].GetIdx] ) ;

        OutChar [PST[i].OutPutIdx++] =
                        NodeChar [NBT[DMATTx].GetIdx++] ;
        }
```

```
 printf ("\n\n") ;

return ;
}
```

D.1.14 TermSystem

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       3  -  TermSystem                *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Performs termination processing *
*    INPUTS:                     NONE                            *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      NONE                            *
*    GLOBAL VARIABLES CHANGED:   NONE                            *
*    GLOBAL TABLES USED:         PST                             *
*    GLOBAL TABLES CHANGED:      NONE                            *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             X.4  -  DispTime                *
*                                X.6  -  DispElapsedTime         *
*    CALLING MODULES:            0  -  Main                      *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

TermSystem ()

    {
    int     i ,             /* index into PST */
            *temp ;         /* temporary pointer */

    for ( i = 0; i < NBROFPORTS; i++ )
        {                   /* restore old interrupt settings */
        temp      = PST [ i ].IntVectAddr ;
        *( temp ) = PST [ i ].StorIntVect ;
        *(temp+1) = PST [ i ].StorPSW ;
        }

    printf ("\n\n++++++++  FEP aborted at ") ;

    DispTime () ;

    printf ("- - - -> elapsed time  = ") ;

    DispElapsedTime () ;

    fclose ( fp ) ;

    return ;
    }
```

D.1.15 SOCInterruptServiceRoutine

```
/*****************************************************************
*                                                                *
*  MODULE NUMBER / NAME:      1.2.1 - SOCInterruptServiceRoutine *
*                                                                *
*  DATE:                      3 Nov 83                           *
*  VERSION:                   A1                                 *
*  FUNCTION:                  Controls movement of characters    *
*                             from Receiver Data Buffer (RBUF)   *
*                             to queue InChar.                   *
*  INPUTS:                    NONE                               *
*  OUTPUTS:                   NONE                               *
*  GLOBAL VARIABLES USED:     NONE                               *
*  GLOBAL VARIABLES CHANGED:  AbortFlag                          *
*  GLOBAL TABLES USED:        PST                                *
*  GLOBAL TABLES CHANGED:     InChar                             *
*                             PST                                *
*  FILES READ:                NONE                               *
*  FILES WRITTEN:             NONE                               *
*  MODULES CALLED:            NONE                               *
*  CALLING MODULES:           1.2 - InitInterrupts (activation)  *
*                                                                *
*  AUTHOR:                    Capt Allan F. Masty, GCS-83D       *
*  HISTORY:                   VSN  A1  -   3 November 1983       *
*                                                                *
*****************************************************************/

SOCInterruptServiceRoutine ()

    {
    entint () ;          /* save registers */

    InChar [PST[SOC].InPutIdx] = *PST[SOC].RcvDataAddr ;
    *PST[SOC].TxmDataAddr      = *PST[SOC].RcvDataAddr ;

    if ( InChar [ PST [SOC].InPutIdx ] == CR )
        *PST [SOC].TxmDataAddr = LF ;

    switch ( InChar [ PST [SOC].InPutIdx ] )
        {
        case    CTRLC :
            {
            AbortFlag = YES ;
            break ;
            }

        case    DEL :
            {
            if ( PST [SOC].InPutIdx > PST [SOC].InGetIdx )
                {
```

```
                --PST [SOC].InPutIdx ;
                *PST [SOC].TxmDataAddr = BACKSPACE ;
                while ((*PST[SOC].TxmStatAddr & 0200) == 0);
                *PST [SOC].TxmDataAddr = SPACE ;
                while ((*PST[SOC].TxmStatAddr & 0200) == 0);
                *PST [SOC].TxmDataAddr = BACKSPACE ;
                }
            break ;
            }

        default :
            {
            if (++PST [SOC].InPutIdx == PST [SOC].InHighIdx )
                --PST [SOC].InPutIdx ;
            break ;
            }
        }

retint () ;          /* restore registers */

return ;
}
```

D.1.16 T1InterruptServiceRoutine

```
/*****************************************************************
*                                                                *
*  MODULE NUMBER / NAME:      1.2.2 - T1InterruptServiceRoutine  *
*                                                                *
*  DATE:                      3 Nov 83                           *
*  VERSION:                   A1                                 *
*  FUNCTION:                  Controls movement of characters    *
*                             from Receiver Data Buffer (RBUF)   *
*                             to queue InChar.                   *
*  INPUTS:                    NONE                               *
*  OUTPUTS:                   NONE                               *
*  GLOBAL VARIABLES USED:     NONE                               *
*  GLOBAL VARIABLES CHANGED:  NONE                               *
*  GLOBAL TABLES USED:        PST                                *
*  GLOBAL TABLES CHANGED:     InChar                             *
*                             PST                                *
*  FILES READ:                NONE                               *
*  FILES WRITTEN:             NONE                               *
*  MODULES CALLED:            NONE                               *
*  CALLING MODULES:           1.2 - InitInterrupts (activation)  *
*                                                                *
*  AUTHOR:                    Capt Allan F. Masty, GCS-83D       *
*  HISTORY:                   VSN  A1  -   3 November 1983       *
*                                                                *
*****************************************************************/

T1InterruptServiceRoutine ()

    {
    entint () ;          /* save registers */

    InChar [PST[T1].InPutIdx] = *PST[T1].RcvDataAddr ;
    *PST[T1].TxmDataAddr      = *PST[T1].RcvDataAddr ;

    if ( InChar [ PST [T1].InPutIdx ] == CR )
        *PST [T1].TxmDataAddr = LF ;

    if ( InChar [ PST [T1].InPutIdx ] == DEL )
        {
        if ( PST [T1].InPutIdx > PST [T1].InGetIdx )
            {
            --PST [T1].InPutIdx ;
            *PST [T1].TxmDataAddr = BACKSPACE ;
            while ((*PST[T1].TxmStatAddr & 0200) == 0);
            *PST [T1].TxmDataAddr = SPACE ;
            while ((*PST[T1].TxmStatAddr & 0200) == 0);
            *PST [T1].TxmDataAddr = BACKSPACE ;
            }
        }
```

```
else
    if (++PST [T1].InPutIdx == PST [T1].InHighIdx )
        --PST [T1].InPutIdx ;

retint () ;          /* restore registers */

return ;
}
```

D.1.17 T2InterruptServiceRoutine

```
/*****************************************************************
*                                                                *
*  MODULE NUMBER / NAME:       1.2.3 - T2InterruptServiceRoutine *
*                                                                *
*  DATE:                       3 Nov 83                          *
*  VERSION:                    A1                                *
*  FUNCTION:                   Controls movement of characters   *
*                              from Receiver Data Buffer (RBUF)  *
*                              to queue InChar.                  *
*  INPUTS:                     NONE                              *
*  OUTPUTS:                    NONE                              *
*  GLOBAL VARIABLES USED:      NONE                              *
*  GLOBAL VARIABLES CHANGED:   NONE                              *
*  GLOBAL TABLES USED:         PST                               *
*  GLOBAL TABLES CHANGED:      InChar                            *
*                              PST                               *
*  FILES READ:                 NONE                              *
*  FILES WRITTEN:              NONE                              *
*  MODULES CALLED:             NONE                              *
*  CALLING MODULES:            1.2 - InitInterrupts (activation) *
*                                                                *
*  AUTHOR:                     Capt Allan F. Masty, GCS-83D      *
*  HISTORY:                    VSN  A1  -   3 November 1983      *
*                                                                *
*****************************************************************/

T2InterruptServiceRoutine ()

    {
    entint () ;          /* save registers */

    InChar [PST[T2].InPutIdx] = *PST[T2].RcvDataAddr ;
    *PST[T2].TxmDataAddr      = *PST[T2].RcvDataAddr ;

    if ( InChar [ PST [T2].InPutIdx ] == CR )
        *PST [T2].TxmDataAddr = LF ;

    if ( InChar [ PST [T2].InPutIdx ] == DEL )
        {
        if ( PST [T2].InPutIdx > PST [T2].InGetIdx )
            {
            --PST [T2].InPutIdx ;
            *PST [T2].TxmDataAddr = BACKSPACE ;
            while ((*PST[T2].TxmStatAddr & 0200) == 0);
            *PST [T2].TxmDataAddr = SPACE ;
            while ((*PST[T2].TxmStatAddr & 0200) == 0);
            *PST [T2].TxmDataAddr = BACKSPACE ;
            }
        }
```

```
else
    if (++PST [T2].InPutIdx == PST [T2].InHighIdx )
        --PST [T2].InPutIdx ;

retint () ;          /* restore registers */

return ;
}
```

D.1.18 T3InterruptServiceRoutine

```
/*****************************************************************
*                                                                *
*  MODULE NUMBER / NAME:      1.2.4 - T3InterruptServiceRoutine  *
*                                                                *
*  DATE:                      3 Nov 83                           *
*  VERSION:                   A1                                 *
*  FUNCTION:                  Controls movement of characters    *
*                             from Receiver Data Buffer (RBUF)   *
*                             to queue InChar.                   *
*  INPUTS:                    NONE                               *
*  OUTPUTS:                   NONE                               *
*  GLOBAL VARIABLES USED:     NONE                               *
*  GLOBAL VARIABLES CHANGED:  NONE                               *
*  GLOBAL TABLES USED:        PST                                *
*  GLOBAL TABLES CHANGED:     InChar                             *
*                             PST                                *
*  FILES READ:                NONE                               *
*  FILES WRITTEN:             NONE                               *
*  MODULES CALLED:            NONE                               *
*  CALLING MODULES:           1.2 - InitInterrupts (activation)  *
*                                                                *
*  AUTHOR:                    Capt Allan F. Masty, GCS-83D       *
*  HISTORY:                   VSN  A1  -   3 November 1983       *
*                                                                *
*****************************************************************/

T3InterruptServiceRoutine ()

    {
    entint () ;          /* save registers */

    InChar [PST[T3].InPutIdx] = *PST[T3].RcvDataAddr ;
    *PST[T3].TxmDataAddr      = *PST[T3].RcvDataAddr ;

    if ( InChar [ PST [T3].InPutIdx ] == CR )
        *PST [T3].TxmDataAddr = LF ;

    if ( InChar [ PST [T3].InPutIdx ] == DEL )
        {
        if ( PST [T3].InPutIdx > PST [T3].InGetIdx )
            {
            --PST [T3].InPutIdx ;
            *PST [T3].TxmDataAddr = BACKSPACE ;
            while ((*PST[T3].TxmStatAddr & 0200) == 0);
            *PST [T3].TxmDataAddr = SPACE ;
            while ((*PST[T3].TxmStatAddr & 0200) == 0);
            *PST [T3].TxmDataAddr = BACKSPACE ;
            }
        }
```

```
else
    if ( ++PST [T3].InPutIdx > PST [T3].InHighIdx )
        --PST [T3].InPutIdx ;

retint () ;          /* restore registers */

return ;
}
```

D.1.19 T4InterruptServiceRoutine

```
/*****************************************************************
*                                                                *
*  MODULE NUMBER / NAME:        1.2.5 - T4InterruptServiceRoutine *
*                                                                *
*  DATE:                        3 Nov 83                         *
*  VERSION:                     A1                               *
*  FUNCTION:                    Controls movement of characters  *
*                               from Receiver Data Buffer (RBUF) *
*                               to queue InChar.                 *
*  INPUTS:                      NONE                             *
*  OUTPUTS:                     NONE                             *
*  GLOBAL VARIABLES USED:       NONE                             *
*  GLOBAL VARIABLES CHANGED:    NONE                             *
*  GLOBAL TABLES USED:          PST                              *
*  GLOBAL TABLES CHANGED:       InChar                           *
*                               PST                              *
*  FILES READ:                  NONE                             *
*  FILES WRITTEN:               NONE                             *
*  MODULES CALLED:              NONE                             *
*  CALLING MODULES:             1.2 - InitInterrupts (activation)*
*                                                                *
*  AUTHOR:                      Capt Allan F. Masty, GCS-83D     *
*  HISTORY:                     VSN  A1  -   3 November 1983     *
*                                                                *
*****************************************************************/

T4InterruptServiceRoutine ()

    {
    entint () ;          /* save registers */

    InChar [PST[T4].InPutIdx] = *PST[T4].RcvDataAddr ;
    *PST[T4].TxmDataAddr      = *PST[T4].RcvDataAddr ;

    if ( InChar [ PST [T4].InPutIdx ] == CR )
        *PST [T4].TxmDataAddr = LF ;

    if ( InChar [ PST [T4].InPutIdx ] == DEL )
        {
        if ( PST [T4].InPutIdx > PST [T4].InGetIdx )
            {
            --PST [T4].InPutIdx ;
            *PST [T4].TxmDataAddr = BACKSPACE ;
            while ((*PST[T4].TxmStatAddr & 0200) == 0);
            *PST [T4].TxmDataAddr = SPACE ;
            while ((*PST[T4].TxmStatAddr & 0200) == 0);
            *PST [T4].TxmDataAddr = BACKSPACE ;
            }
        }
```

```
else
    if ( ++PST [T4].InPutIdx > PST [T4].InHighIdx )
        --PST [T4].InPutIdx ;

retint () ;          /* restore registers */

return ;
}
```

D.1.20 T5InterruptServiceRoutine

```
/*****************************************************************
*                                                                *
*  MODULE NUMBER / NAME:        1.2.6 - T5InterruptServiceRoutine *
*                                                                *
*  DATE:                        3 Nov 83                          *
*  VERSION:                     A1                                *
*  FUNCTION:                    Controls movement of characters   *
*                               from Receiver Data Buffer (RBUF)  *
*                               to queue InChar.                  *
*  INPUTS:                      NONE                              *
*  OUTPUTS:                     NONE                              *
*  GLOBAL VARIABLES USED:       NONE                              *
*  GLOBAL VARIABLES CHANGED:    NONE                              *
*  GLOBAL TABLES USED:          PST                               *
*  GLOBAL TABLES CHANGED:       InChar                            *
*                               PST                               *
*  FILES READ:                  NONE                              *
*  FILES WRITTEN:               NONE                              *
*  MODULES CALLED:              NONE                              *
*  CALLING MODULES:             1.2 - InitInterrupts (activation) *
*                                                                *
*  AUTHOR:                      Capt Allan F. Masty, GCS-83D      *
*  HISTORY:                     VSN  A1  -   3 November 1983      *
*                                                                *
*****************************************************************/

T5InterruptServiceRoutine ()

    {
    entint () ;          /* save registers */

    InChar [PST[T5].InPutIdx] = *PST[T5].RcvDataAddr ;
    *PST[T5].TxmDataAddr      = *PST[T5].RcvDataAddr ;

    if ( InChar [ PST [T5].InPutIdx ] == CR )
        *PST [T5].TxmDataAddr = LF ;

    if ( InChar [ PST [T5].InPutIdx ] == DEL )
        {
        if ( PST [T5].InPutIdx > PST [T5].InGetIdx )
            {
            --PST [T5].InPutIdx ;
            *PST [T5].TxmDataAddr = BACKSPACE ;
            while ((*PST[T5].TxmStatAddr & 0200) == 0);
            *PST [T5].TxmDataAddr = SPACE ;
            while ((*PST[T5].TxmStatAddr & 0200) == 0);
            *PST [T5].TxmDataAddr = BACKSPACE ;
            }
        }
```

```
else
    if ( ++PST [T5].InPutIdx > PST [T5].InHighIdx )
        --PST [T5].InPutIdx ;

retint () ;          /* restore registers */

return ;
}
```

D.1.21 T6InterruptServiceRoutine

```
/*****************************************************************
*                                                                *
*  MODULE NUMBER / NAME:        1.2.7 - T6InterruptServiceRoutine *
*                                                                *
*  DATE:                        3 Nov 83                         *
*  VERSION:                     A1                               *
*  FUNCTION:                    Controls movement of characters  *
*                               from Receiver Data Buffer (RBUF) *
*                               to queue InChar.                 *
*  INPUTS:                      NONE                             *
*  OUTPUTS:                     NONE                             *
*  GLOBAL VARIABLES USED:       NONE                             *
*  GLOBAL VARIABLES CHANGED:    NONE                             *
*  GLOBAL TABLES USED:          PST                              *
*  GLOBAL TABLES CHANGED:       InChar                           *
*                               PST                              *
*  FILES READ:                  NONE                             *
*  FILES WRITTEN:               NONE                             *
*  MODULES CALLED:              NONE                             *
*  CALLING MODULES:             1.2 - InitInterrupts (activation)*
*                                                                *
*  AUTHOR:                      Capt Allan F. Masty, GCS-83D     *
*  HISTORY:                     VSN  A1  -   3 November 1983     *
*                                                                *
*****************************************************************/

T6InterruptServiceRoutine ()

    {
    entint () ;          /* save registers */

    InChar [PST[T6].InPutIdx] = *PST[T6].RcvDataAddr ;
    *PST[T6].TxmDataAddr      = *PST[T6].RcvDataAddr ;

    if ( InChar [ PST [T6].InPutIdx ] == CR )
        *PST [T6].TxmDataAddr = LF ;

    if ( InChar [ PST [T6].InPutIdx ] == DEL )
        {
        if ( PST [T6].InPutIdx > PST [T6].InGetIdx )
            {
            --PST [T6].InPutIdx ;
            *PST [T6].TxmDataAddr = BACKSPACE ;
            while ((*PST[T6].TxmStatAddr & 0200) == 0);
            *PST [T6].TxmDataAddr = SPACE ;
            while ((*PST[T6].TxmStatAddr & 0200) == 0);
            *PST [T6].TxmDataAddr = BACKSPACE ;
            }
        }
```

```
else
    if ( ++PST [T6].InPutIdx > PST [T6].InHighIdx )
        --PST [T6].InPutIdx ;

retint () ;          /* restore registers */

return ;
}
```

D.1.22 T7InterruptServiceRoutine

```
/*****************************************************************
*                                                                *
*  MODULE NUMBER / NAME:      1.2.8 - T7InterruptServiceRoutine  *
*                                                                *
*  DATE:                      3 Nov 83                           *
*  VERSION:                   A1                                 *
*  FUNCTION:                  Controls movement of characters    *
*                             from Receiver Data Buffer (RBUF)   *
*                             to queue InChar.                   *
*  INPUTS:                    NONE                               *
*  OUTPUTS:                   NONE                               *
*  GLOBAL VARIABLES USED:     NONE                               *
*  GLOBAL VARIABLES CHANGED:  NONE                               *
*  GLOBAL TABLES USED:        PST                                *
*  GLOBAL TABLES CHANGED:     InChar                             *
*                             PST                                *
*  FILES READ:                NONE                               *
*  FILES WRITTEN:             NONE                               *
*  MODULES CALLED:            NONE                               *
*  CALLING MODULES:           1.2 - InitInterrupts (activation)  *
*                                                                *
*  AUTHOR:                    Capt Allan F. Masty, GCS-83D       *
*  HISTORY:                   VSN  A1  -   3 November 1983       *
*                                                                *
*****************************************************************/

T7InterruptServiceRoutine ()

    {
    entint () ;          /* save registers */

    InChar [PST[T7].InPutIdx] = *PST[T7].RcvDataAddr ;
    *PST[T7].TxmDataAddr      = *PST[T7].RcvDataAddr ;

    if ( InChar [ PST [T7].InPutIdx ] == CR )
        *PST [T7].TxmDataAddr = LF ;

    if ( InChar [ PST [T7].InPutIdx ] == DEL )
        {
        if ( PST [T7].InPutIdx > PST [T7].InGetIdx )
            {
            --PST [T7].InPutIdx ;
            *PST [T7].TxmDataAddr = BACKSPACE ;
            while ((*PST[T7].TxmStatAddr & 0200) == 0);
            *PST [T7].TxmDataAddr = SPACE ;
            while ((*PST[T7].TxmStatAddr & 0200) == 0);
            *PST [T7].TxmDataAddr = BACKSPACE ;
            }
        }
```

```
else
    if ( ++PST [T7].InPutIdx > PST [T7].InHighIdx )
        --PST [T7].InPutIdx ;

retint () ;          /* restore registers */

return ;
}
```

D.1.23 DMAInterruptServiceRoutine

```
/*****************************************************************
*                                                                *
*  MODULE NUMBER / NAME:      1.2.9 - DMAInterruptServiceRoutin  *
*                                                                *
*  DATE:                      3 Nov 83                           *
*  VERSION:                   A1                                 *
*  FUNCTION:                  Controls movement of characters    *
*                             from the DMA interface to NodeChar *
*  INPUTS:                    NONE                               *
*  OUTPUTS:                   NONE                               *
*  GLOBAL VARIABLES USED:     DMABusyFlag                        *
*                             DMAwc                              *
*  GLOBAL VARIABLES CHANGED:  DMABusyFlag                        *
*                             DMAwc                              *
*  GLOBAL TABLES USED:        NBT                                *
*                             NodeChar                           *
*                             PST                                *
*  GLOBAL TABLES CHANGED:     NBT                                *
*                             PST                                *
*  FILES READ:                NONE                               *
*  FILES WRITTEN:             NONE                               *
*  MODULES CALLED:            1.2.9.1  -  SetUpForInputDMA       *
*  CALLING MODULES:           1.2 - InitInterrupts (activation)  *
*                                                                *
*  AUTHOR:                    Capt Allan F. Masty, GCS-83D       *
*  HISTORY:                   VSN  A1  -   3 November 1983       *
*                                                                *
*****************************************************************/

DMAInterruptServiceRoutine ()

    {
    entint () ;                    /* save registers */

    printf ("\n ... DMACSR = %o \n", *DMACSR ) ;

    if ( *DMACSR & DMANEX )        /* if NEX memory accessed */
        printf ("\n . . . . NEX at %o", *DMABAR ) ;
    else
        {
        if ( *DMACSR & DMAIREQUEST )  /* host input request */
            SetUpForInputDMA () ;
        else
            {
            printf ("\n . . . . Output Transfer Complete\n\n") ;
            switch  DMABusyFlag
                {
                case  1 :              /* Input word expected */
                    {
```

```
                    printf ("\n DMABusyFlag = %d\n",DMABusyFlag);
                    DMAwc = *DMADBR ;
                    if ((NBT[DMATTx].PutIdx+(DMAwc*2))
                                        <= NBT[DMATTx].HighIdx)
                        {
                        ++DMABusyFlag ; /* block input expected */
                        *DMABAR  = &(NodeChar[NBT[DMATTx].PutIdx]);
                        *DMAWCR  = -DMAwc ;
                        *DMACSR |=  DMAODIRECTION ;
                        *DMACSR &= ~DMAOMODE ;
                        *DMACSR |=  DMAGO ;
                        }
                    else
                        printf ("\n +=+= DMAwc = %d , PutIdx = %d \n
                                    DMAwc, NBT[DMATTx].PutIdx );
                    break ;
                    }

                case  2 :       /* Input block expected */
                    {
                    printf ("\n DMABusyFlag = %d\n",DMABusyFlag);
                    break ;
                    }

                case  3 :       /* Output word in progress */
                    {
                    printf ("\n DMABusyFlag = %d\n",DMABusyFlag);
                    break ;
                    }

                case  4 :       /* Output block in progress */
                    {
                    printf ("\n DMABusyFlag = %d\n",DMABusyFlag);
                    break ;
                    }

                default :
                    {
                    printf ("\n DMABusyFlag = %d\n",DMABusyFlag);
                    break ;
                    }

                }               /* end of 'switch' processing */
            }
        }
    retint () ;         /* restore registers */

    return ;
    }
```

D.1.24 SetUpForInputDMA

```
/*****************************************************************
*                                                                *
*  MODULE NUMBER / NAME:        1.2.9.1  -  SetUpForInputDMA     *
*                                                                *
*  DATE:                        3 Nov 83                         *
*  VERSION:                     A1                               *
*  FUNCTION:                    Programs the DMA interface to    *
*                               receive a transfer from Host.    *
*  INPUTS:                      NONE                             *
*  OUTPUTS:                     NONE                             *
*  GLOBAL VARIABLES USED:       NONE                             *
*  GLOBAL VARIABLES CHANGED:    DMABusyFlag                      *
*  GLOBAL TABLES USED:          InChar                           *
*                               PST                              *
*  GLOBAL TABLES CHANGED:       PST                              *
*  FILES READ:                  NONE                             *
*  FILES WRITTEN:               NONE                             *
*  MODULES CALLED:              NONE                             *
*  CALLING MODULES:             1.2.9 - DMAInterruptServiceRoutine*
*                                                                *
*  AUTHOR:                      Capt Allan F. Masty, GCS-83D     *
*  HISTORY:                     VSN A1 -  3 November 1983        *
*                                                                *
*****************************************************************/

SetUpForInputDMA ()

    {
    printf ("\n . . DMA Input Interrupt Request") ;

    DMABusyFlag = 1 ;          /* Input word expected */
    *DMABAR     = &( InChar [PST [DMA].InPutIdx++] ) ;

    if ( *DMACSR & DMAIMODE )  /* if request for 'word' mode */
        {
        *DMACSR |= DMAOMODE ;      /* set output 'word' mode  */
        *DMAWCR  = -1 ;            /* set Word Count register */
        }
    else                    /* request is for 'block' transfer */
        {
        *DMACSR &= ~( DMAOMODE ); /* set output 'block' mode */
        *DMAWCR  = -( DMAwc ) ;   /* set Word Count register */
        DMABusyFlag++ ;           /* Input block expected    */
        }

    printf ("\n ... DMACSR = %o \n", *DMACSR ) ;

    return ;
    }
```

D.2 LFEPHI.C Program modules

```
/*****************************************************************
*                                                                *
*    TITLE:              LSI FEP Extended Memory 'C' Program     *
*    FILENAME:           LFEPHI.C                                *
*                                                                *
*    DATE:               3 Nov 83                                *
*    VERSION:            A1                                      *
*    OWNER:              Capt Allan F. Masty                     *
*    COMPUTER SYSTEM:    LSI-11/23                               *
*    OPERATING SYSTEM:   RT11XM                                  *
*    LANGUAGE:           Telecon 'C'                             *
*                                                                *
*    CONTENTS:           X.1  -  DispPST                         *
*                        X.2  -  DispNBT                         *
*                        X.3  -  GetCurrentTime                  *
*                        X.4  -  DispTime                        *
*                        X.5  -  CalcElapsedTime                 *
*                        X.6  -  DispElapsedTime                 *
*                        X.7  -  GatherStats                     *
*                                                                *
*    FUNCTION:           Provides an extended memory resident    *
*                        collection of service subroutines for   *
*                        use by the low memory LFEPLO.C program  *
*                                                                *
*****************************************************************/


        /* ................  GLOBALS  ................... */

#include        "lfepio.h"      /*   Standard IO routines   */

extern  int     fopen () ;

/* . . .  gtime ()  subr call data structure support   . . . */

struct  timerec
            {
            int w1 ;    /* high order byte = minutes,
                               low order byte =   hours  */
            int w2 ;    /* high order byte = tics,
                               low order byte = seconds  */
            }
        time ;

char    tim [11] ;

int
        StartHr  = -1 ,
```

```
        StartMin = -1 ,
        StartSec = -1 ,
        StartTic = -1 ,

          hours ,
        minutes ,
        seconds ,
          ticks ;

extern  int     *fp ;

/*  . .  Port Status ( PS ) Table defines & declarations  . .  */

#define    NBROFPORTS    9      /* NBROFTERMINALS +SOC +DMA */

struct  PortStatusRecord
        {
        char    TID [4] ;        /* VT-100 Terminal ID      */
        char    TermMode ;       /* line or character mode  */

        int     InLowIdx   ;     /* Input buffer pointers   */
        int     InPutIdx   ;
        int     InGetIdx   ;
        int     InHighIdx  ;

        int     OutLowIdx  ;     /* Output buffer pointers  */
        int     OutPutIdx  ;
        int     OutGetIdx  ;
        int     OutHighIdx ;

        int     *RcvStatAddr ;   /* receive port status addr */
        int     *RcvDataAddr ;   /* receive port data addr   */
        int     *TxmStatAddr ;   /* transmit port status addr*/
        int     *TxmDataAddr ;   /* transmit port data addr  */

        int     *IntVectAddr ;   /* receive port int. addr   */
        int     IntRoutineAddr ; /* Interrupt service routine*/

        int     StorIntVect  ;
        int     StorPSW      ;
        } ;

#define   PSRSIZE  sizeof ( PortStatusRecord )


/* . . . . .  misc defines for Node Buffer table  . . . . . */

#define     NBROFBUFFERS    2   /* Nmbr of Node buffers   */
```

```
/*  . . . . . Node Buffer Table declaration  . . . .  */

struct  NodeBufferRecord
            {
            char    OrgNode [4] ;
            int     LowIdx  ;
            int     PutIdx  ;
            int     GetIdx  ;
            int     HighIdx ;
            } ;

#define   NBRSIZE  sizeof ( NodeBufferRecord )


/*     . . . .  Message Transport Layer Header  . . .   */


struct  MsgTransportLayerHeader
            {
            char    TID [3] ;
            char    Mode ;
            char    MsgSeqNbr [4] ;
            char    MsgCharCnt [4] ;
            char    OrgNode [3] ;
            } ;

#define   THTSIZE  sizeof ( MsgTransportLayerHeader )
```

D.2.1 DispPST

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       X.1  -  DispPST                 *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Displays (upon the SOC terminal *
*                                screen) certain PST values.     *
*    INPUTS:                     P = pointer to PST              *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      NONE                            *
*    GLOBAL VARIABLES CHANGED:   NONE                            *
*    GLOBAL TABLES USED:         PST                             *
*    GLOBAL TABLES CHANGED:      NONE                            *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             NONE                            *
*    CALLING MODULES:            2.1.1  -  EvalSOCInput          *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

DispPST ( P )

    struct  PortStatusDataRecord  *P ;

    {

    int         i ;             /* loop control variable */

    printf ("\n i  TID  M  I.LOW I.PUT I.GET I.HIGH") ;
    printf ("  O.LOW O.PUT O.GET O.HIGH\n\n") ;

    for ( i = 0; i < NBROFPORTS ;  i++ )
        {
        printf (" %d",  i ) ;
        printf ("  %s", P-> TID ) ;
        printf ("  %c", P-> TermMode ) ;

        if ( P-> InLowIdx < 1000 )
            printf (" ") ;
        if ( P-> InLowIdx <  100 )
            printf (" ") ;
        if ( P-> InLowIdx <   10 )
            printf (" ") ;
        printf ("  %d", P-> InLowIdx ) ;
```

```
if ( P-> InPutIdx < 1000 )
    printf (" ") ;
if ( P-> InPutIdx <  100 )
    printf (" ") ;
if ( P-> InPutIdx <   10 )
    printf (" ") ;
printf ("  %d", P-> InPutIdx ) ;

if ( P-> InGetIdx < 1000 )
    printf (" ") ;
if ( P-> InGetIdx <  100 )
    printf (" ") ;
if ( P-> InGetIdx <   10 )
    printf (" ") ;
printf ("  %d", P-> InGetIdx ) ;

if ( P-> InHighIdx < 1000 )
    printf (" ") ;
if ( P-> InHighIdx <  100 )
    printf (" ") ;
if ( P-> InHighIdx <   10 )
    printf (" ") ;
printf ("   %d",        P-> InHighIdx ) ;

if ( P-> OutLowIdx < 1000 )
    printf (" ") ;
if ( P-> OutLowIdx <  100 )
    printf (" ") ;
if ( P-> OutLowIdx <   10 )
    printf (" ") ;
printf ("   %d", P-> OutLowIdx ) ;

if ( P-> OutPutIdx < 1000 )
    printf (" ") ;
if ( P-> OutPutIdx <  100 )
    printf (" ") ;
if ( P-> OutPutIdx <   10 )
    printf (" ") ;
printf ("  %d", P-> OutPutIdx ) ;

if ( P-> OutGetIdx < 1000 )
    printf (" ") ;
if ( P-> OutGetIdx <  100 )
    printf (" ") ;
if ( P-> OutGetIdx <   10 )
    printf (" ") ;
printf ("  %d", P-> OutGetIdx ) ;

if ( P-> OutHighIdx < 1000 )
    printf (" ") ;
if ( P-> OutHighIdx <  100 )
```

```
            printf (" ") ;
        if ( P-> OutHighIdx <    10 )
            printf (" ") ;
        printf ("   %d", P-> OutHighIdx ) ;

        printf ("\n") ;

        P += PSRSIZE ;

        }

return  ;

}
```

D.2.2 DispNBT

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       X.2  -  DispNBT                 *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Displays (upon the SOC terminal *
*                                screen) certain NBT values.     *
*    INPUTS:                     P = pointer to NBT              *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      NONE                            *
*    GLOBAL VARIABLES CHANGED:   NONE                            *
*    GLOBAL TABLES USED:         NBT                             *
*    GLOBAL TABLES CHANGED:      NONE                            *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             NONE                            *
*    CALLING MODULES:            2.1.1  -  EvalSOCInput          *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

DispNBT ( P )

    struct   NodeBufferRecord   *P ;
    {
    int          n ;            /* loop control variable */

    printf ("\n i   low    put    get   high  \n\n" ) ;

    for ( n = 0; n < ( NBROFBUFFERS ); n++ )
        {
        printf (" %d", n ) ;

        if ( P-> LowIdx  < 10000 )
            printf (" ") ;
        if ( P-> LowIdx  <  1000 )
            printf (" ") ;
        if ( P-> LowIdx  <   100 )
            printf (" ") ;
        if ( P-> LowIdx  <    10 )
            printf (" ") ;
        printf ("  %d", P-> LowIdx  ) ;

        if ( P-> PutIdx  < 10000 )
            printf (" ") ;
        if ( P-> PutIdx  <  1000 )
```

```
            printf (" ") ;
        if ( P-> PutIdx  <   100 )
            printf (" ") ;
        if ( P-> PutIdx  <    10 )
            printf (" ") ;
        printf ("  %d", P-> PutIdx  ) ;

        if ( P-> GetIdx  < 10000 )
            printf (" ") ;
        if ( P-> GetIdx  <  1000 )
            printf (" ") ;
        if ( P-> GetIdx  <   100 )
            printf (" ") ;
        if ( P-> GetIdx  <    10 )
            printf (" ") ;
        printf ("  %d", P-> GetIdx  ) ;

        if ( P-> HighIdx  < 10000 )
            printf (" ") ;
        if ( P-> HighIdx  <  1000 )
            printf (" ") ;
        if ( P-> HighIdx  <   100 )
            printf (" ") ;
        if ( P-> HighIdx  <    10 )
            printf (" ") ;
        printf ("  %d\n", P-> HighIdx ) ;

        P += NBRSIZE ;
        }
    return  ;
    }
```

D.2.3 GetCurrentTime

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       X.3  -  GetCurrentTime          *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Evokes Macro-11 coded function  *
*                                "gtime" (located in library     *
*                                LFMLLO.MAC) to read system time *
*    INPUTS:                     NONE                            *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      hours, minutes, seconds, ticks  *
*    GLOBAL VARIABLES CHANGED:   hours, minutes, seconds, ticks, *
*                                StartHr, StartMin, StartSec,    *
*                                StartTic.                       *
*    GLOBAL TABLES USED:         time                            *
*    GLOBAL TABLES CHANGED:      time, tim                       *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             NONE                            *
*    CALLING MODULES:            X.4  -  DispTime                *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

GetCurrentTime ()

    {
    int         k ;

    gtime ( time ) ;

    hours   =   time.w1         & 0377  ;
    minutes = ( time.w1 >> 8 ) & 0377 ;
    seconds =   time.w2         & 0377  ;
    ticks   = ( time.w2 >> 8 ) & 0377 ;

    if ( StartHr < 0 )
        {
        StartHr  =   hours ;
        StartMin = minutes ;
        StartSec = seconds ;
        StartTic =   ticks ;
        }

    tim [ 0] = (  hours / 10 ) + '0' ;
    tim [ 1] = (  hours % 10 ) + '0' ;
```

```
tim [ 2] = ':' ;
tim [ 3] = (minutes / 10 ) + '0' ;
tim [ 4] = (minutes % 10 ) + '0' ;
tim [ 5] = ':' ;
tim [ 6] = (seconds / 10 ) + '0' ;
tim [ 7] = (seconds % 10 ) + '0' ;
tim [ 8] = ':' ;
tim [ 9] = (  ticks / 10 ) + '0' ;
tim [10] = (  ticks % 10 ) + '0' ;

return ;

}
```

D.2.4 DispTime

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:      X.4  -  DispTime                 *
*                                                                *
*    DATE:                      3 Nov 83                         *
*    VERSION:                   A1                               *
*    FUNCTION:                  Displays (upon the SOC terminal  *
*                               screen) current system time.     *
*    INPUTS:                    NONE                             *
*    OUTPUTS:                   NONE                             *
*    GLOBAL VARIABLES USED:     NONE                             *
*    GLOBAL VARIABLES CHANGED:  NONE                             *
*    GLOBAL TABLES USED:        tim                              *
*    GLOBAL TABLES CHANGED:     NONE                             *
*    FILES READ:                NONE                             *
*    FILES WRITTEN:             NONE                             *
*    MODULES CALLED:            X.3   -  GetCurrentTime          *
*    CALLING MODULES:           2     -  PerfNormalActivities    *
*                               2.1.1 -  EvalSOCInput            *
*                               3     -  TermSystem              *
*                                                                *
*    AUTHOR:                    Capt Allan F. Masty, GCS-83D     *
*    HISTORY:                   VSN A1 -  3 November 1983        *
*                                                                *
*****************************************************************/

DispTime ()

    {
    int         k ;

    GetCurrentTime () ;

    for ( k = 0; k < sizeof (tim); k++ )
        printf ("%c", tim [k] ) ;
    printf ("\n");

    return ;

    }
```

D.2.5 CalcElapsedTime

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       X.5  -  CalcElapsedTime         *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Calculates elapsed time from    *
*                                FEP start-up to FEP termination *
*    INPUTS:                     NONE                            *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      hours, minutes, seconds, ticks, *
*                                StartHr, StartMin, StartSec,    *
*                                StartTic                        *
*    GLOBAL VARIABLES CHANGED:   hours, minutes, seconds, ticks  *
*    GLOBAL TABLES USED:         NONE                            *
*    GLOBAL TABLES CHANGED:      NONE                            *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             X.3  -  GetCurrentTime          *
*    CALLING MODULES:            X.6  -  DispElapsedTime         *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

CalcElapsedTime ()
    {

    int         k ;

    GetCurrentTime () ;

    if ( ( ticks -= StartTic ) < 0 )
        {
        ticks += 60 ;
        --seconds ;
        }

    if ( ( seconds -= StartSec ) < 0 )
        {
        seconds += 60 ;
        --minutes ;
        }

    if ( ( minutes -= StartMin ) < 0 )
        {
        minutes += 60 ;
        --hours ;
```

```
    }

if ( ( hours -= StartHr ) < 0 )
    hours += 24 ;

return ;
}
```

D.2.6 DispElapsedTime

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       X.6  -  DispElapsedTime         *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Displays (upon the SOC terminal *
*                                screen) the time duration       *
*                                determined in CalcElapsedTime.  *
*    INPUTS:                     NONE                            *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      NONE                            *
*    GLOBAL VARIABLES CHANGED:   NONE                            *
*    GLOBAL TABLES USED:         tim                             *
*    GLOBAL TABLES CHANGED:      tim                             *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              NONE                            *
*    MODULES CALLED:             X.5  -  CalcElapsedTime         *
*    CALLING MODULES:            3    -  TermSystem              *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

DispElapsedTime ()
    {
    int         k ;

    CalcElapsedTime () ;

    tim [ 0] = (  hours / 10 ) + '0' ;
    tim [ 1] = (  hours % 10 ) + '0' ;
    tim [ 2] = ':' ;
    tim [ 3] = (minutes / 10 ) + '0' ;
    tim [ 4] = (minutes % 10 ) + '0' ;
    tim [ 5] = ':' ;
    tim [ 6] = (seconds / 10 ) + '0' ;
    tim [ 7] = (seconds % 10 ) + '0' ;
    tim [ 8] = ':' ;
    tim [ 9] = (  ticks / 10 ) + '0' ;
    tim [10] = (  ticks % 10 ) + '0' ;

    for ( k = 0; k < sizeof (tim); k++ )
        printf ("%c", tim [k] ) ;
    printf ("\n");

    return ;
    }
```

D.2.7 GatherStats

```
/*****************************************************************
*                                                                *
*    MODULE NUMBER / NAME:       X.7  -  GatherStats             *
*                                                                *
*    DATE:                       3 Nov 83                        *
*    VERSION:                    A1                              *
*    FUNCTION:                   Copies node queue accounting    *
*                                information to file LSIFEP.DAT  *
*                                for off-line data reduction.    *
*    INPUTS:                     p      = pointer to char array  *
*                                action = reason for call:       *
*                                                                *
*                                         1 = entry into node    *
*                                         2 = exit from node     *
*    OUTPUTS:                    NONE                            *
*    GLOBAL VARIABLES USED:      NONE                            *
*    GLOBAL VARIABLES CHANGED:   NONE                            *
*    GLOBAL TABLES USED:         tim                             *
*    GLOBAL TABLES CHANGED:      NONE                            *
*    FILES READ:                 NONE                            *
*    FILES WRITTEN:              LSIFEP.DAT                      *
*    MODULES CALLED:             NONE                            *
*    CALLING MODULES:            2.1.2  -  MoveMsgtoNodeTTxDMA   *
*                                2.3.1  -  TTxtoDMAOutput        *
*                                2.3.2  -  DMAtoTTxOutput        *
*                                                                *
*    AUTHOR:                     Capt Allan F. Masty, GCS-83D    *
*    HISTORY:                    VSN A1 -  3 November 1983       *
*                                                                *
*****************************************************************/

GatherStats ( p, action )

      char      p[] ;
       int  action ;
    {
    int          k ;
    char        *c ;

    GetCurrentTime ( ) ;

    for ( k = 0; k < sizeof (tim); k++ )
        putc ( tim [k], fp ) ;

    putc (' ', fp ) ;

    putc ( action + '0', fp ) ;

    putc (' ', fp ) ;
```

```
for ( c = 0; c < THTSIZE; c++ )
    putc (  p [ c ], fp ) ;

putc ( '\015', fp ) ;
putc ( '\012', fp ) ;

return ;

}
```

APPENDIX E

LSI FEP MEMORY LOAD MAP (LSIFEP.MAP)

This is the baseline Memory Load Map for the LSIFEX.SAV executable program image.

```
RT-11 LINK  V06.01B      Load Map          Fri 07-Oct-83 00:00:00
LSIFEX.SAV       Title:  .MAIN.  Ident:  V04.00

Section  Addr    Size    Global  Value   Global  Value

 . ABS.  000000  001000    (RW,I,GBL,ABS,OVR)
                         $SYSV$  000012
 $OHAND  001000  000252    (RW,I,GBL,REL,CON)
                         $OVRHV  001000  $OVRH   001004
                         V$READ  001034  V$DONE  001046
                         $VDF5   001234  $VDF4   001236
                         $VDF1   001246  $VDF2   001250
 $OTABL  001252  000160    (RW,D,GBL,REL,OVR)
         001432  057052    (RW,I,LCL,REL,CON)
                         SHELL   001432  SHELLX  001452
                         NXTARG  002436  SHLERR  003016
                         CLOSTD  003066  GETCHA  003140
                         PUTCHA  003206  GETS    003442
                         FGETS   003706  PUTS    004206
                         FPUTS   004262  FREOPE  004342
                         PRINTF  004374  FPRINT  004424
                         SPRINT  004504  OUTF    004566
                         OUTDEC  005376  OUTOCT  005670
                         STRCAT  005772  STRCMP  006116
                         STRCOM  006256  STRCOP  006416
                         STRCPY  006520  STRLEN  006576
                         ECHAR   006646  ENCHAR  006656
                         GCHAR   006714  INTASC  006756
                         ASCIII  007102  GETTIM  007236
                         AG1FL   007506  ECHO    007510
                         ENABLO  007512  LINE    007514
                         LINEPT  007722  OBPTR   007724
                         OBSIZE  007726  RARY1T  007730
                         STDERR  007732  STDIN   007734
                         STDOUT  007736  UCONLY  007740
                         CCSWIT  007742  CCMPY   010006
                         CCMULT  010006  CCDIV   010114
                         CCASR   010344  CCLRS   010344
                         CCLLS   010366  CCASL   010366
                         SETIDP  010414  ASTFN   010426
                         GETACH  010756  PUTACH  010772
                         INITIA  011006  ENTINT  011022
                         RETINT  011052  GTIME   011076
                         GAREA   011172  EXIT    011176
                         NODECH  014140  MAIN    024000
                         INITSY  024026  INITPS  026246
                         INITIN  030174  PERFNO  030630
                         SRVINP  032276  EVALSO  033066
                         MOVEMS  033466  BLDTRA  034140
                         SRVOUT  034746  SRVNOD  035432
```

```
                                TTXTOD  036556  DMATOT  037356
                                TERMSY  040176  SOCINT  040572
                                T1INTE  041472  T2INTE  041656
                                T3INTE  042520  T4INTE  043362
                                T5INTE  044224  T6INTE  045066
                                T7INTE  045730  DMAINT  046572
                                SETUPF  050000  ABORTF  050350
                                DMABUS  050352  DMAWC   050354
                                ENDIDX  050356  FILEOP  050360
                                FP      050362  GODMAF  050364
                                LASTMS  050366  NBTCHA  050370
                                NBT     050372  PSTCHA  050422
                                PST     050424  STARTI  051414
                                STOPID  051416  THT     051420
                                INCHAR  051464  OUTCHA  055074

 SYS$I    060504 000114    (RW,I,LCL,REL,CON)
                           $DIVTK  060504  $DIV60  060532
                           ISPY    060576  $GVAL   060602

 SYS$S    060620 000004    (RW,D,LCL,REL,CON)
                           $SYSLB  060620  $LOCK   060622
                           $CRASH  060623

 Segment size = 060624 = 12490. words

------------------------------------------------------------

Virtual overlay region  000001
------------------------

Partition       000001  Segment 000001

          100002 006676    (RW,I,LCL,REL,CON)
                           DISPPS@ 100002  DISPNB@ 102540
                           GETCUR  104242  DISPTI@ 105112
                           DISPEL@ 105234  CALCEL  105736
                           GATHER@ 106256  HOURS   106604
                           MINUTE  106606  STARTH  106610
                           STARTM  106612  STARTS  106614
                           STARTT  106616  SECOND  106620
                           TIME    106622  TIM     106626
                           TICKS   106642
 Segment size = 006676 = 1759.  words

Virtual overlay region  000002
------------------------

Partition       000002  Segment 000002

          120002 013576    (RW,I,LCL,REL,CON)
```

```
                                FOPEN @ 120010  FSIZE   120414
                                FDELET  120416  FCLOSE@ 120534
                                GETC   @ 133122  PUTC   @ 133306

 Segment size = 013576 = 3007.  words


Transfer address = 001432,
High limit = 060622 = 12489. words


Virtual high limit = 133576 = 23487. words,
next free address = 140000


Extended memory required = 022500 = 4768.  words
```

APPENDIX F

LSI FEP USER'S GUIDE

This Appendix contains detailed information on the operation of the LSI FEP system. It contains sections describing System Initialization, System Operation, System Termination, and Off-line Accounting Data Reduction. Additional reference for the commands recognized by the RT-11 monitor can be found in the RT-11 documentation set [9].

F.1 System Initialization -

The steps required for system initialization include Booting the RT-11XM Monitor, Setting the Date and Time, and Starting the LSIFEP Program Image.

F.1.1 Booting the RT-11XM Monitor - The operator places the system disk (volume ID = "RT11A") into the leftmost disk drive of the Plessey Peripheral Systems cabinet. This disk drive (DY0:) is known to the system as the "system" disk drive. The rightmost disk drive (DY1:) is known to the system as the "user" disk drive. The operator then keys in the following (upper case) command:

DY<cr>

where <cr> represents the "return" key. This tells the microcode bootstrap program to boot the system from device DY0. When the boot completes, the RT11SJ monitor will be active. To boot the RT11XM monitor from the RT11SJ monitor, type the command:

BOOT RT11XM<cr>

To return to the RT11SJ monitor from the RT11XM monitor, type the command:

BOOT RT11SJ<cr>

At anytime during operation, the operator may desire to reboot the system without powering-down and powering-up the Plessey. This can be accomplished by toggling the "BOOT" switch on the front Plessey panel. This action will result in the microcode bootstrap loader outputting an asterisk on the system console screen. At this point, the

above boot command can again be given. The contents of disk volume ID "RT11A" are contained in Figure F-1.

F.1.2 Setting the Date and Time - When the system is first booted, no date will be known to the monitor. To remind the operator to set the date, the start-up indirect command files for both monitors ("STARTS.COM and STARTX.COM) contain a request to the operating system to display the current date. This command:

DATE<cr>

will result in the monitor responding with the message:

?KMON-W-No Date

At this point, the operator should set the date by an appropriate command:

DATE 16-OCT-83<cr>

Time can then be set. First, ensure that the "LTC" switch mounted on the front Plessey panel is in the "ON" position. This switch controls the Line Time Clock interrupt hardware feature. Then, issue the appropriate command:

TIME 10:45:30<cr>

```
 16-Oct-83
 Volume ID: RT11A
 Owner    : Masty

PIP    .SAV   23  09-Sep-80     DUP    .SAV   41  09-Sep-80
DIR    .SAV   17  09-Sep-80     FORMAT.SAV   19  09-Sep-80
RESORC.SAV    15  09-Sep-80     SYSMAC.SML   42  09-Sep-80
MACRO  .SAV   51  09-Sep-80     CREF   .SAV    6  09-Sep-80
LINK   .SAV   41  09-Sep-80     SRCCOM.SAV   13  09-Sep-80
ODT    .OBJ    9  21-Feb-80     QUEUE  .REL   14  09-Sep-80
BATCH  .SAV   26  21-Feb-80     BAX    .SYS    7  04-Aug-83
DXX    .SYS    4  04-Aug-83     DYX    .SYS    4  04-Aug-83
DMX    .SYS    5  04-Aug-83     MTX    .SYS    9  04-Aug-83
LPX    .SYS    2  23-Dec-81     LSX    .SYS    2  23-Dec-81
CRX    .SYS    3  04-Aug-83     NLX    .SYS    2  04-Aug-83
RT11XM.SYS  102  04-Aug-83     SWAP   .SYS   25  09-Sep-80
TT     .SYS    2  09-Sep-80     DY     .SYS    4  09-Sep-80
RT11SJ.SYS   67  09-Sep-80     STARTS.BAK    1  02-Aug-83
KED    .SAV   60  09-Sep-80     RUNOFF.SAV   33  10-Aug-81
DUMP   .SAV    8  09-Sep-80     LIBR   .SAV   22  31-Mar-81
SYSLIB.OBJ   47  13-Jul-81     QUEMAN.SAV   13  21-Feb-80
ERROUT.SAV   17  21-Feb-80     CC     .SAV   95  02-Nov-82
STARTS.COM    1  26-Aug-83     STARTX.BAK    1  16-Sep-83
STARTX.COM    1  16-Sep-83

 39 Files, 854 Blocks
 120 Free blocks
```

Fig. F-1 DY0: Directory

F.1.3 Starting the LSIFEP Program Image - Disk (volume ID = "LSI FEP 2") contains the LSI FEP program. The contents of this disk are contained in Figure F-2. This disk must be mounted in the DY1 drive and the following command issued:

RUN LSIFEP<cr>

The 'C' Shell responds with a prompt:

++

in response to which the operator will key a carriage return. Following this last carriage return, the 'C' Shell releases control to the "main" program module in the LSIFEP.SAV program. After program initialization has completed, the message:

++++++++ FEP Activated at 10:46:15

will be displayed upon the system operator console.

LSIFEP.SAV is a program image which can be run using the RT11SJ or RT11XM monitor. The LSI FEP "2 disk also contains the LSIFEX.SAV program image, which can only be run with the RT11XM monitor.

```
 16-Oct-83
 Volume ID: LSI FEP #1
 Owner    : Masty

D       .COM    1   28-Sep-83     F       .COM    1   28-Sep-83
X       .COM    2   28-Sep-83     C       .COM    5   28-Sep-83
LSIFEX.MAP      7   07-Oct-83     CH5     .RNO    1   03-Oct-83
CH6     .RNO    1   03-Oct-83     FIG11   .RNO    4   03-Oct-83
FIG31   .RNO    4   03-Oct-83     FIG32   .RNO    2   04-Oct-83
TABLE1.RNO     14   04-Oct-83     FIG21   .RNO    3   04-Oct-83
A1      .RNO   25   03-Oct-83     TABLE2.RNO      4   03-Oct-83
A6      .RNO    1   03-Oct-83     A7      .RNO    3   03-Oct-83
A8      .RNO    1   03-Oct-83     FIG41   .RNO    6   03-Oct-83
B       .COM    1   03-Oct-83     CH4     .RNO   32   07-Oct-83
CH3     .RNO   40   12-Oct-83     LFEPLO.C       47   12-Oct-83
BIB     .RNO   11   07-Oct-83     A5      .RNO    8   07-Oct-83
CH2     .RNO   74   12-Oct-83     A3      .RNO   68   07-Oct-83
THESIS.RNO      3   12-Oct-83     A2      .RNO   23   07-Oct-83
CH1     .RNO   30   12-Oct-83     A4      .RNO   50   12-Oct-83

 30 Files, 472 Blocks
 502 Free blocks
```

Fig. F-2 DY1: Directory

F.2 System Operation -

The LSI FEP system is expected to function with no required operator control or maintenance activity. Three commands have been implemented for operator monitoring of the system status. These commands are described in the following paragraphs.

F.2.1 NBT - The status of the Node Buffer Table can be displayed by issuing the command:

NBT<cr>

from the SOC terminal to the LSI FEP program. The format of a typical Node Buffer Table is contained in Figure F-3.

F.2.2 PST - The status of the Port Status Table can be displayed by issuing the command:

PST<cr>

from the SOC terminal to the LSI FEP program. The format of a typical Port Status Table is contained in Figure F-4.

F.2.3 TIME - The current system time can be displayed by issuing the command:

TIME<cr>

from the SOC terminal to the LSI FEP program.

i	low	put	get	high
0	0	0	0	1999
1	2000	2000	2000	3999

LEGEND:

i = index into NBT
0 = TTxDMA node
1 = DMATTx node

low = NBT [i].LowIdx
put = NBT [i].PutIdx
get = NBT [i].GetIdx
high = NBT [i].HighIdx

Fig. F-3 Node Buffer Table (NBT)

i	TID	M	I.LOW	I.PUT	I.GET	I.HIGH	O.LOW	O.PUT	O.GET	O.HIGH
0	SOC	L	0	4	0	199	0	0	0	199
1	T.1	L	200	200	200	399	200	200	200	399
2	T.2	L	400	400	400	599	400	400	400	599
3	T.3	L	600	600	600	799	600	600	600	799
4	T.4	L	800	800	800	999	800	800	800	999
5	T.5	L	1000	1000	1000	1199	1000	1000	1000	1199
6	T.6	L	1200	1200	1200	1399	1200	1200	1200	1399
7	T.7	L	1400	1400	1400	1599	1400	1400	1400	1599
8	DMA	L	1600	1600	1600	1799	1600	1600	1600	1799

LEGEND:

i = index into PST
TID = Terminal identification

M = Terminal Mode
'L' = Line
'C' = Character

I.LOW = PST [i].InLowIdx (InChar index)
I.PUT = PST [i].InPutIdx
I.GET = PST [i].InGetIdx
I.HIGH = PST [i].InHighIdx
O.LOW = PST [i].OutLowIdx (OutChar index)
O.PUT = PST [i].OutPutIdx
O.GET = PST [i].OutGetIdx
O.HIGH = PST [i].OutHighIdx

Fig. F-4 Port Status Table (PST)

F.3 System Termination -

The system can be terminated in an orderly manner by issuing a control-C (C) from the SOC terminal. The "CTRL" key is held down while the "C" key is depressed. This results in the LSIFEP program restoring interrupt vector addresses and closing the LSIFEP.DAT file prior to returning control to the monitor.

F.4 Off-line Accounting Data Reduction -

The LSIFEP.DAT file is created during each run of the LSIFEP.SAV program. This file contains the accounting information describing the data movement through the network nodes. Each line in the file contains the time of recording, a movement code, and the message Transport Header. A typical LSIFEP.DAT file appears in Figure F-5. Although the raw data file is produced, time did not allow the generation of a data reduction program which could process this file.

```
10:16:29:18 1 T.1L00020018TTx
10:16:29:21 2 T.1L00020018TTx
10:16:30:58 1 SOCL00030020DMA
10:16:31:01 2 SOCL00030020DMA
10:16:37:27 1 T.5L00040020TTx
10:16:37:30 2 T.5L00040020TTx
10:16:40:21 1 SOCL00050018TTx
10:16:40:24 2 SOCL00050018TTx
10:16:44:32 1 T.7L00060058TTx
10:16:44:34 2 T.7L00060058TTx


    LEGEND:

record format:   HH:MM:SS:TT R TID M NUMB COUN NOD

     HH = hours       (2 chars)
     MM = minutes     (2 chars)
     SS = seconds     (2 chars)
     TT = ticks       (2 chars)

     R = reason for statistics gathering     (1 char)
             1 = entry into queue
             2 =  exit from queue

     TID = Terminal Identifier        (3 chars)

     M = Terminal mode                (1 char )

     NUMB = Message Sequence Number   (4 chars)

     COUN = Message Character Count   (4 chars)

     NOD = Node queue ID              (3 chars)
```

Fig. F-5 LSIFEP.DAT File Format

APPENDIX G

LSI FEP PROGRAMMER'S GUIDE

This appendix provides software documentation for the LSI FEP maintenance programmer. It begins by describing the LSI-11 source code edit facility -- the Keypad Editor [13]. It then proceeds with descriptions of the compilation, assembly, and linking processes. Amplifying information can be found in Chapter 4, Appendix C, and Appendix D.

G.1 Editing The Source Code -

The PDP-11 Keypad Editor (KED) was used to create the source programs during this implementation. KED is used on a DEC VT-100 terminal and requires the file 'KED.SAV' on

the DY0: disk (Fig F-1). KED allows use of the terminal's alphanumeric keys as well as the small keypad. The keypad keys activate programmed requests for cursor positioning, text deletion, text string search, and other functions described in DEC documentation [13].

Activation of KED is accomplished by the command:

EDIT LFEPLO.C<cr>

where <cr> represents striking the RETURN key. After this command, any of the KED directives [13] can be issued.

G.2 Compiling The Source Code -

The Telecon 'C' compiler was used to compile the 'C' language source code into RT-11 Macro-11 assembly instructions. The file "CC.SAV" must exist on the DY0: drive [fig F-1]. The 'C' compiler is activated by the command:

R CC<cr>

at which time the 'C' shell prompt appears:

++

The programmer then keys the input ('C' source) file name

and the output (Macro-11) file name [21; 23]. The format is as follows:

LFEPLO.C >LFEPLO.MAC<cr>

All diagnostics from the compiler are directed to the display terminal screen. The macro file which is produced as output may be several times the size (disk blocks) of the input 'C' text file. To ensure a large contiguous disk area for the macro file, it is sometimes necessary to "SQUEEZE" the DY1: disk prior to invoking the 'C' compiler. The format for the squeeze command [9:4.167] is as follows:

SQUEEZE/NOQUERY DY1:<cr>

G.3 Assembling The Macro File -

The PDP-11 Macro Assembler [15] was used to produce the object modules. Input to the assembler consists of macro files containing the PDP-11 assembly instructions [20:53-162]. The format [9:4.128] of the macro command is:

MACRO LFEPLO<cr>

This command specifies an input file of "LFEPLO.MAC" (.MAC

is default file type) and an output file of "LFEPLO.OBJ" (default file type is .OBJ while default file name is same as the input file name).

G.4 Linking The Object Modules -

So far, the documentation has only addressed the programming issues for one program (LFEPLO.C) through edit, compile, and assembly. Any functions and data items that could not be found by the compiler during compilation were treated as external references which would be supplied later. The macro assembler, likewise, deferred action on these externals and passed on the object file with these externals still undefined. The linker, however, requires all references to be resolved before it can create the .SAV executable program image file. Therefore, it is at this stage that all the pieces [para 4.3] of the LSIFEP (or LSIFEX) system are brought together. Two indirect command files [9:4.9] have been created to accomplish the linking function. These indirect command files exist on the DY1: disk [fig F-2] as "F.COM" (creates "LSIFEP.SAV") and "X.COM" (creates "LSIFEX.SAV"). When all the ".OBJ" files have been created, the "F.COM" file can be invoked using the command:

@F<cr>

The RT-11 monitor then executes the commands within the "F.COM" file. The file contents are shown in Figure G-1 for "F.COM" and Figure G-2 for "X.COM". The output of the "F.COM" file consists of the "LSIFEP.SAV" executable image and a text file "LSIFEP.MAP" which shows the memory mapping produced by the linker. The contents of "LSIFEX.MAP" is contained in Appendix E. Execution instructions for the executable images are contained in Appendix F.

```
! This indirect command file is F.COM  ---
! It is used to invoke th RT-11 Linker.
!
! F.COM is invoked by typing     @F
!      at the console keyboard.
!
! The following files must be defined
!      prior to invoking the linker:
!
!           LFEPIO.OBJ
!           LFMLLO.OBJ
!           LFMLHI.OBJ
!            NBUFF.OBJ
!           LFEPHI.OBJ
!           LFEPLO.OBJ
!            TBUFF.OBJ
!
! The linker creates two files:
!
!           LSIFEP.SAV
!           LSIFEP.MAP

r link
lsifep,lsifep=lfepio,lfmllo,lfmlhi,nbuff//
lfephi,lfeplo,tbuff//
^C
```

Fig. G-1 'F.COM' Indirect Command File

```
! This indirect command file is X.COM  ---
! It is used to invoke the RT-11 Linker.
!
! X.COM is invoked by typing      @X
!       at the console keyboard.
!
! The following files must be defined
!       prior to invoking X.COM:
!
!
! The commands forwarded to the Linker will create
!       an extended memory LSIFEX.SAV file which
!       can be run on an LSI-11/23 using the
!       RT11XM extended memory monitor.
!
!
!   LFEPIO.OBJ      -       linked to  low memory
!   LFMLLO.OBJ      -       linked to  low memory
!   LFMLHI.OBJ      -       linked to high memory
!    NBUFF.OBJ      -       linked to  low memory
!   LFEPHI.OBJ      -       linked to high memory
!   LFEPLO.OBJ      -       linked to  low memory
!    TBUFF.OBJ      -       linked to  low memory
!
!
! The linker creates two files:
!
!           LSIFEX.SAV
!           LSIFEX.MAP

r link
lsifex,lsifex=lfepio,lfmllo,nbuff,lfeplo,tbuff//
lfephi/V:1
lfmlhi/V:2//
^C
```

Fig. G-2 'X.COM' Indirect Command File

VITA

Capt Allan F. Masty was born in Detroit, Michigan on 13 February 1949. He graduated from St. Martin HS, Detroit, in 1967. He received a Bachelor of Science degree in Physics from the University of Detroit in 1971. He then taught high school mathematics and physics for three years prior to entering Air Force Officer Training School in 1974. Upon commissioning, he was assigned as a Space Surveillance Officer at Detactment 7, 14th Missile Warning Squadron (ADCOM), MacDill AFB, Florida. During this tour, he cross-trained from the space operations career field into the computer programming career field. He was then assigned to the first PAVE PAWS SLBM Early Warning unit --- 6th Missile Warning Squadron, Otis AFB, Massachusetts --- in 1978. He then transferred to the second PAVE PAWS site --- 7th Missile Warning Squadron, Beale AFB, California --- in 1979. While at Beale AFB, he performed duties as the Chief, Tactical Applications Branch for the PAVE PAWS System Programming Agency (SPA), Stategic Air Command. He entered the Air Force Institute of Technology in 1982 to pursue a Masters degree in Computer Science.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/EE/83D-13	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) see box 19	
12. PERSONAL AUTHOR(S) Masty, Allan Floyd B.S., Capt, USAF	
13a. TYPE OF REPORT MS Thesis 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1983 December 15. PAGE COUNT 265
16. SUPPLEMENTARY NOTATION	Approved for public release: IAW AFR 190-17 LYNN E. WOLAVER Dean for Research and Professional Development Air Force Institute of Technology (AIC) Wright-Patterson AFB OH (45433)
17. COSATI CODES FIELD: 09 / GROUP: 02 / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Communications Front End Processor, DEC LSI-11/23, Terminal Concentrator

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

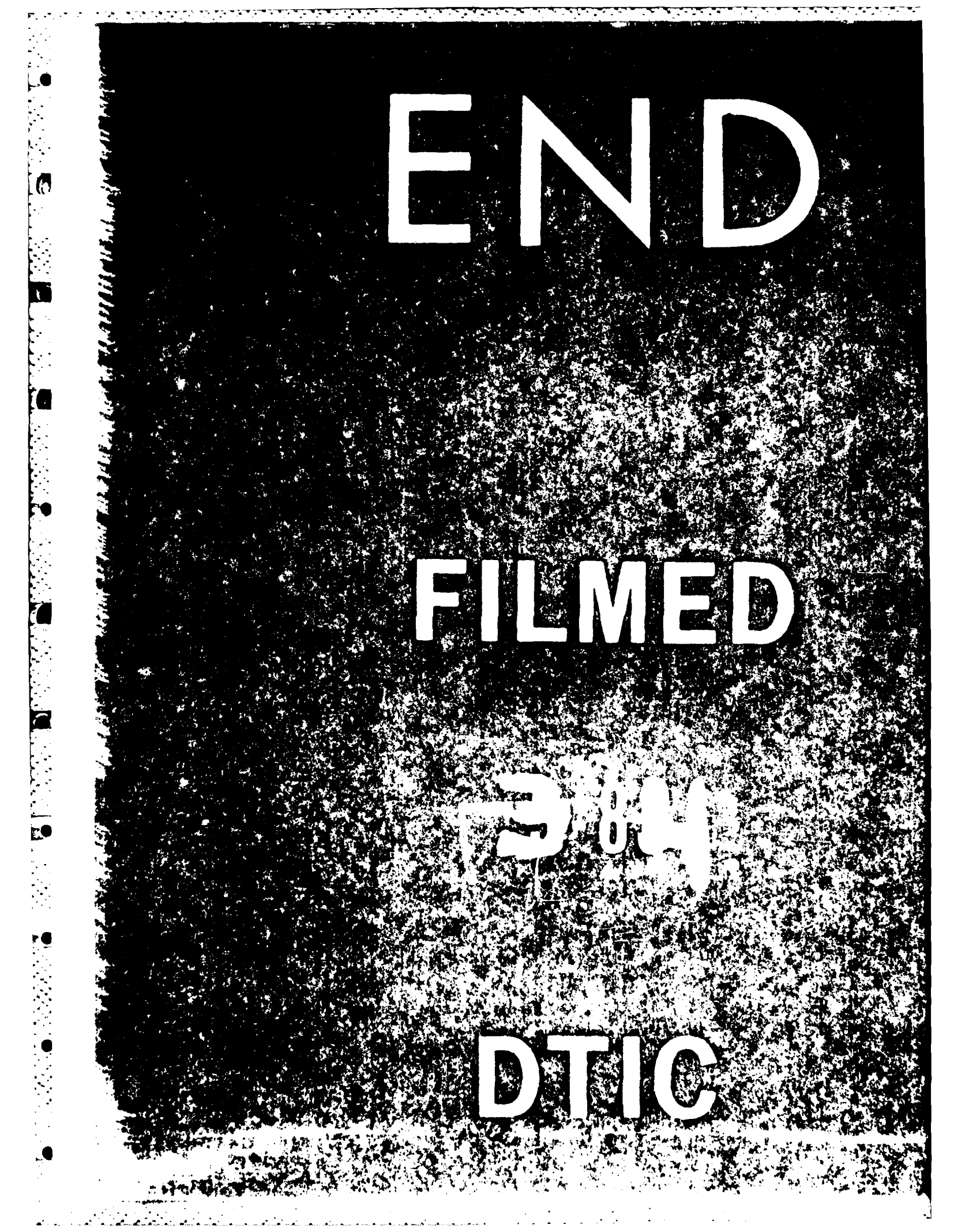
Title: Development of a Communications Front End Processor (FEP) for the VAX-11/780 Using an LSI-11/23.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ CLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Gary Lamont, Ph.D.	22b. TELEPHONE NUMBER (Include Area Code) 513-255-3576 22c. OFFICE SYMBOL AFIT/ENG

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

A Communications Front-End Processor (FEP) was implemented for a Digital Equipment Corporation (DEC) VAX-11/780 using a DEC LSI-11/23 microcomputer. The LSI-11/23 serviced eight DEC VT-100 terminals and communicated with the VAX-11/780 over an Able Computer Technology, Inc Direct Memory Access (DMA) interface. This investigation proceeded from a FEP design provided in a previous work and culminated in the Telecon 'C' compiler language coding of those design specifications. The design was translated into structure charts defining software module functions and interfaces. Program Design Language (PDL) was then used to outline the processing steps in a structured programming format for each software module. A data dictionary was constructed to document the data and functional module interfaces. The code was implemented in a "top-down" manner.


END
FILMED
DTIC